NORTHWESTERN UNIVERSITY

Sensing and Delivery of Biomolecules with Spherical Nucleic Acid Nanoparticle Conjugates

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Interdisciplinary Biological Sciences

By

Isaac Nathaniel Larkin

EVANSTON, ILLINOIS

September 2020

ABSTRACT

Sensing and Delivery of Biomolecules with Spherical Nucleic Acid Nanoparticle Conjugates

Over the past fifty years, techniques for synthesizing and manipulating matter on the 1-100 nanometer scale have led to the development of nanoparticle-based approaches to both disease diagnosis and treatment. The modification of nanoparticles with biological macromolecules such as proteins and nucleic acids has led to the development of highly sensitive detectors of disease biomarkers and has facilitated the delivery of therapeutic molecules into target tissues and cells. Challenges and opportunities in the development of biomolecule-functionalized nanoparticles include: (1) how to balance the sensitivity of a biomarker diagnostic assay against the cost and complexity of the equipment required to perform the assay; (2) developing design rules for the construction of nucleic-acid-based biosensors of small molecules; and (3) cytosolic delivery of extracellular enzymes by avoiding endosomal entrapment. Present in this thesis is an exploration of biomolecule-functionalized nanoparticles for addressing these challenges. Chapter one surveys the origins and applications of a wide variety of bio-functionalized nanoparticles. In particular, I focus on spherical nucleic acids (SNAs), nanoparticles densely functionalized with a highly oriented oligonucleotide shell, which possess structural and biological properties distinct from linear nucleic acids that have enabled the development of biosensors and new therapeutics. Chapter two details the use of antibody-functionalized gold nanoparticles to develop a dual readout assay for device-free and highly sensitive detection of an anthrax biomarker. Chapter three explores the properties and design rules of aptamer NanoFlares, a class of SNA functionalized with DNA aptamers and hybridized with small fluorophore-modified oligonucleotides, in the context of work aimed at developing an assay for small molecule biomarkers of human stress. Chapter four investigates the possibilities and challenges of adapting SNAs for the delivery of gene-editing enzymes, with the goal of using gene editing to better understand the

endosomal escape of SNAs. Finally, chapter five discusses the challenges encountered during these projects and provides perspective on how researchers could build on this work going forward.

Thesis Advisor: Professor Chad A. Mirkin

ACKNOWLEDGMENTS

This dissertation exists because of the help and support of an entire village's worth of people. I owe every one of them a debt of gratitude and will try to thank them here.

I first thank my thesis advisor, Chad Mirkin, for welcoming me into a lab and a research group that I have learned so much from, for pushing me to be the most rigorous, clear-minded and clear-spoken scientist I can be, and for giving me the time and resources to pursue research projects that excited me, even when they didn't always pan out. I would like to thank my thesis committee, Rich Carthew, Keith Tyo and Mike Jewett, for giving me advice when I truly needed it. I'd like to thank Carole Labonne, Jason Brickner, and Josh Leonard, who in different ways are the reason I joined the IBIS program and had opportunity to become part of the Northwestern University research community.

I also thank the entire Mirkin office staff, Elizabeth, Pam, Tanushri and Sara, for their kindness, their willingness and their bottomless capacity to help. I also thank Cathy Prullage, who cares for all the IBiS students and without whom I certainly would have forgotten to register for the quarter on numerous occasions.

I thank all of my peers in Mirkin lab, who are some of the smartest, most motivated and most helpful people I've ever met. Thank you to Hang, Gokay, Zhu, Kacper, Shuya, Caroline, Sasha and so many others, for your collaboration and your company these past few years. Thank you as well to my collaborators at the Air Force Research Lab, Peter, Jorge, BJ, Stephanie, and Monica, who made flying out to Ohio a rewarding and intellectually stimulating experience.

Thank you to Northwestern's synthetic biology community, from the professors who helped attract me to Northwestern in the first place, to the friends and collaborators I've learned from and look forward to talking to. Perhaps the most valuable thing I have gained from my time as a graduate student is a deep understanding of the problems I care about and the scientific, technological and other solutions I'm passionate about helping to build. I learned these things through my conversations and the work we've shared together. Joe, Weston, Sarah, Jordan, Adam, you were the highlights of my PhD.

Finally, I am deeply grateful to my family, especially my parents, Eric and Kathi, for raising me and encouraging my love of the living world, for believing me and supporting me my whole life. I love you. To my sisters, for being excellent sisters. I love you. To Galia, for welcoming me into your family and helping to make Chicago feel more like home. I love you, and I will always be ready and willing to help fix your electronics.

And to my wife, Alida. What can I say? You are everything to me, even more so these past few years. I don't express often enough how lucky I feel to have met you, to be your partner, to have you as mine. Thank you for your companionship, your wit, your kindness and your patience, and your support in so many things. I truly would not be here, writing the acknowledgments on a finished dissertation, without you. I can't wait to build the rest of our lives together.

TABLE OF CONTENTS

Contents

ABSTRACT..……..……3

ACKNOWLEDGMENTS……..……5

TABLE OF CONTENTS……..……7

CHAPTER 1: Introduction……..……..11

1.1 Summary……...…….…….…….……12

1.2 Nanoparticles in Diagnostics…….…….……..…12

1.3 Nanoparticles in Therapeutics…….…….……14

1.4 Spherical Nucleic Acids……..…..……16

1.5 SNAs in Diagnostics……...…….……18

1.6 SNAs in Therapeutics…….…….……..…20

CHAPTER 2: Dual Readout Assay for Device-Free and Sensitive Anthrax Biomarker Detection……24

2.1 Introduction…….……..25

2.2 Design of Assay…….……...27

2.3 Screen for Anti-PA_{83} Monoclonal Antibody Sandwich…….……..28

2.4 Colorimetric PA_{83} Detection in Phosphate Buffered Saline…….……31

2.5 PA_{83} Detection in Human Serum…….……32

2.6 Scanometric PA_{83} Detection…….……33

2.7 Materials and Methods…….……34

2.7.1 Reagents…….……..…34

2.7.2 Buffered Solutions…….……...…35

2.7.3 Synthesis of Antibody-Coated Gold Nanoparticles…….……..…35

2.7.4 Characterization of AuNP-mAbs…….……..…35

2.7.5 Making Antibody-Coated Wells. …….……36

2.7.6 Screening for Antibody Sandwich Pairs……..……..36

2.7.7 Colorimetric PA_{83} Detection Curves and Kinetics …….……37

2.7.8 Serum PA_{83} Detection Curve…….……...37

2.7.9 Scanometric PA_{83} Detection…….……37

2.7.10 Monoclonal antibody orientation in PA_{83} sandwich……...…38

2.7.11 Optimization of H_2O_2 concentration in TMB/H_2O_2 solution. ……38

2.7.12 Optimization of Pt reduction time. ……..……39

2.7.13 Optimization of incubation times. …….……..…39

2.7.14 ELISA. …….……..39

2.7.15 Blind serum PA_{83} detection…….……40

CHAPTER 3: Modeling, Measuring and Screening Aptamer NanoFlares Toward Detection of Human Stress Biomarkers…….……..41

3.1 Introduction…….……...…42

3.2 Initial Aptamer NanoFlare design…….……45

3.3 DIS11th_3T Truncated Aptamer NanoFlare Design…….……...…46

3.4 Structural Response of DIS11th_3 and DIS11th_3T to DHEA-S...…….……47

3.5 Backfilling Aptamer NanoFlares to Reduce Nonspecific Quenching…..……..…47

3.6 Measuring Flare Hybridization Efficiency…….……..50

3.7 Increasing Flare Strand Length…….……53

3.8 DHEA-S Detection and Batch-To-Batch Variability…….……...55

3.9 Conformational Selection Model of Aptamer NanoFlares…….……...…56

3.10 Determining Aptamer-Flare and Aptamer-DHEA-S Dissociation Constants…..……..57

3.11 Predictions of Equilibrium Conformational Selection Model…….……..58

3.12 Induced Fit Kinetic Model of Aptamer NanoFlares…….……...…61

3.13 Measuring Aptamer-Flare-DHEA-S Binding Kinetics with Bio-Layer Interferometry……65

3.14 Effect of Flare Sequence and DHEA-S on Aptamer-Flare Binding Kinetics…….67

3.15 Suitability of SNAs for Bio-Layer Interferometry……..70

3.16 Cortisol Aptamer-Flare Binding Kinetics……..……..71

3.17 Microarray Screens for Aptamer-Flare Pairs…….……72

3.18 Towards Discovery of Target-Responsive Aptamer-Flare Pairs…….……...77

3.19 Materials and Methods…….……...79

3.19.1 Materials…….……79

3.19.2 Aptamer NanoFlare Synthesis…….……...79

3.19.3 Nuclear Magnetic Resonance of Aptamers…….……80

3.19.4 Fluorescence Measurements…….……..80

3.19.5 Isothermal Titration Calorimetry…….……...80

3.19.6 Bio-Layer Interferometry…….……...80

3.19.7 Microarray Screens…….……81

CHAPTER 4: Exploring the Limits of Cytosolic Enzyme Delivery with CRISPR SNAs……..…82

4.1 Introduction…….……..…83

4.2 Potential SNA-Mediated CRISPR Delivery Methods…….……..…89

4.3 Exploring Cas9/sgRNA Attachment to the Surface of SNAs……...…90

4.4 Exploring Direct Modification of Cas9 to Form CRISPR proSNAs……...92

4.5 Cas9/sgRNA Encapsulation in Liposomal CRISPR SNAs……..95

4.6 Liposomal CRISPR SNAs in Cells……100

4.7 Materials and Methods…….……...…102

4.7.1 Materials…….……102

4.7.2 Cas9 Labeling and Quantification…….……102

4.7.3 Cas9 Ribonucleoprotein Synthesis and Concentration……...…103

4.7.4 Synthesis and Purification of CRISPR SNAs…….……103

4.7.5 Quantification of Cas9 and DNA Loading in Liposomal SNAs……..…103

4.7.6 *In Vitro* Cas9 DNA Cleavage Assay..……..……..…104

4.7.7 Protease Stability Studies…….……..……...…104

4.7.8 Cell Uptake Studies…….……105

4.7.9 Quantification of Gene Editing…….……..105

CHAPTER 5: Outlook, Future Directions…….……106

5.1 Dual Readout Sandwich Immunoassay…….……107

5.2 Aptamer NanoFlares…….……..108

5.3 CRISPR SNAs…….……...…111

REFERENCES…….……...…114

APPENDIX A: Supporting Tables for Chapter 3…….……..…135

APPENDIX B: Supporting Tables for Chapter 4…….……137

APPENDIX C: Supporting Code for Chapter 3…….……141

C.1 Conformational Selection Equilibrium Model Markdown File……141

C.2 Induced Fit and Conformational Selection Kinetic Model Markdown File……..……150

C.3 Microarray Data Processing Markdown File……174

C.4 Representative Microarray Data Analysis Markdown File……185

APPENDIX D: Supporting Code for Chapter 4……225

D.1 Script for Quantifying Gene Editing……...…225

CHAPTER 1: Introduction

1.1 Summary

In both diagnostic assays and therapeutics, nanoparticles display properties distinct from those of their molecular-scale counterparts, and can serve as multifunctional platforms for the attachment and combination of molecular functional groups.[1] These properties enable nanoparticles to detect biomarkers of disease sensitively and with multiple readout modalities, as well as to enter cells and serve as delivery vehicles for therapeutic biomolecules. I present in this thesis an extensive exploration of biomolecule-functionalized nanoparticles for the diagnosis and treatment of disease. I will first discuss the use of antibody-functionalized gold nanoparticles to develop a highly sensitive and specific dual readout assay for an anthrax biomarker. I will then investigate the properties and design rules of aptamer NanoFlares, gold nanoparticles functionalized with DNA aptamers and hybridized with small fluorophore-modified oligonucleotides, in the context of work towards an assay for human stress hormones. Finally, I will explore the possibilities and challenges of oligonucleotide-functionalized liposomes (liposomal spherical nucleic acids) for the delivery of gene-editing enzymes.

1.2 Nanoparticles in Diagnostics

One of the key insights of nanotechnology is that the properties of a nanoparticle can be dramatically different than the properties of a larger mass of the same material. Gold nanoparticles (AuNPs), for instance, display resonant waves of oscillating electrons at the boundary between the nanoparticle's surface and the medium in which it is dispersed.[2] When confined to the surface area of 5-100 nm AuNPs, this surface plasmon resonance (SPR) causes AuNPs to absorb light with extraordinary efficiency, with the wavelengths of peak absorption dictated by the size and shape of the particle.[3] Similarly, nanoparticles of semiconductors such as cadmium sulfide (CdS) have energy levels in the conductance and valence orbital bands that are quantized rather than continuous, with the size of the energy gap between the two bands dictated largely by the size of the nanoparticle.[4] These quantum dots display robust fluorescence emission in a narrow range of wavelengths corresponding to the size of the band gap, meaning that particles with the same material

composition but of different sizes will emit different wavelengths of light upon irradiation with ultraviolet light.

The unique intrinsic properties of some nanoparticles have been harnessed for diagnostic assays and measurements. For instance, the local SPR on the surface of 10-30 nanometer AuNPs gives dispersed solutions of these particles a deep red color.[5] However, aggregation of gold nanoparticles in solution causes a coupling of the particles' electron oscillations, a redshift in the SPR absorption peak, and a change of solution color to blue.[5] This aggregation-induced color change has been used as a visual readout in diagnostics for proteins, nucleic acids and small molecules.[6] Similarly, the narrow emission band and high photostability of quantum dots has led to their development in multiplex diagnostics, with different analytes detected with quantum dots of different colors.[7] Iron oxide nanoparticles, which unlike bulk iron oxide are superparamagnetic, have been developed as contrast agents for magnetic resonance imaging.[8]

Through both covalent chemical conjugation and noncovalent adsorption, nanoparticles can serve as a platform or substrate for the attachment of bio-recognition elements that bind to disease biomarkers of interest.[9] Indeed, most nanoparticles employed in diagnostic assays are functionalized with biomarker-binding biomolecules, including antibodies[10] and other epitope-binding proteins,[11] DNA and RNA oligonucleotides,[12] as well as nucleic acid[13] and peptide[14] aptamers. Researchers have used biomolecule-functionalized nanoparticles to detect, among other things, biomarkers of cancers and infectious diseases.[15, 16, 17, 18, 19] Many of these assays employ the intrinsic properties of the nanoparticles (e.g. visible color for AuNPs and fluorescence for quantum dots) for detection, but others functionalize the nanoparticles with signal generating moieties in addition to the bio-recognition elements. Nanoparticles have also been functionalized with diverse signal-generating moieties, including enzymes like horseradish peroxidase,[20] which can catalyze the conversion of a chemical substrate to a colorful species; catalytic metals such as platinum, which can perform some of the same chemical reactions as enzymes;[21] nanoparticles encapsulating europium (III) ions, which are highly fluorescent;[22] and organic fluorophores,[23] which can be quenched by proximity to AuNPs' surface plasmon resonance and activated by displacement away from

the nanoparticle. Finally, nanoparticles in diagnostic assays are frequently modified with passivating agents that minimize non-specific binding of biomolecules or cells to the nanoparticle surface and increase the particles' colloidal stability. Passivating agents can be biomolecules, such as bovine serum albumin;[24] or they can be synthetic polymers and oligomers, such as thiol-modified polyethylene glycol (PEG).[25]

1.3 Nanoparticles in Therapeutics

The unique properties of nanoparticles, as well as their ability to serve as an attachment platform for multiple functional groups, has inspired research and development of nanoparticle-based therapeutics. As with diagnostics, many nanoparticles have intrinsic properties that enable therapeutic applications. For instance, the energy from the light that an AuNP absorbs so efficiently is released from the particles as infrared, which can heat the solution liquid surrounding the AuNP to temperatures that kill cells.[26, 27] Researchers have investigated harnessing this phenomenon to perform photothermal therapy, or localized heat-based killing by shining tissue-penetrating red light onto tumors suffused with AuNPs.[28] Superparamagnetic iron oxide nanoparticles similarly heat their surrounding solution when subjected to an oscillating magnetic field, and have been used to treat tumors with magnetically-mediated thermal ablation.[29]

As in diagnostics, the modification of nanoparticles with multiple functional groups is key to their utility as therapeutics. Importantly, for many therapeutic applications the functional groups are encapsulated inside polymeric or biomolecular nanoparticles, in addition to being attached to the surface of the nanoparticles. One widely used delivery format is the liposome, a nanoparticle made of at least one phospholipid bilayer, which can carry small molecule and biomolecule drugs in its core.[30]

Nanoparticle functional groups can serve as passivating, targeting, and therapeutic agents. Passivating agents, like chemically conjugated PEG molecules, reduce nonspecific binding or adsorption of biomolecules and cells to the nanoparticles. Such passivation can improve the pharmacokinetic properties of the nanoparticles in the bloodstream, increasing circulation time and thereby increasing the fraction of nanoparticles that reach the target tissue.[31] By both reducing immunogenicity and decreasing the fraction

of therapeutic delivered to off-target tissues, passivation can also minimize the toxicity and increase the therapeutic index of a drug. For example, the first FDA-approve nanomedicine was Doxil®, a PEGylated liposome filled with the chemotherapeutic doxorubicin that delivers a higher dose of drug to tumors with lower side effects than free doxorubicin.[79]

Targeting agents increase the efficiency with which nanoparticle therapeutics deliver their drug cargo to the disease tissue, and reduce the rate of delivery to off-target tissues. Importantly, targeting of nanoparticle therapeutics can be achieved both through the composition of the nanoparticle core, and through the attachment of functional groups to the nanoparticle surface. Nanoparticle size plays a role in tissue targeting: particles smaller than 10 nm are rapidly filtered and cleared by the kidneys,[76] and particles significantly larger than 100 nm are rapidly enveloped and cleared from the bloodstream by phagocytic immune cells in the liver, lungs and spleen.[33] Nanoparticles in the 10-100 nm size range tend to have longer circulation half-lives, before ending up primarily in the liver, spleen, and lungs.[34] In rodent models, particles of this size also tend to accumulate in tissues where capillary walls are leaky, like tumors.[35] The surface charge of a nanoparticle also affects its targeting; nanoparticles with cationic surface charges tend to have increased intracellular delivery (as well as cytotoxicity) compared to particles with neutral or anionic surface charges.[34] Furthermore, both surface charge and nanoparticle core composition can play a role in the intracellular fate of nanoparticle therapeutics: particles with cationic surfaces or cationic core components, such as liposomes doped with the cationic phospholipids dioleoylphosphatidylethanolamine (DOPE) and 1,2-dioleoyl-3-trimethylammoniumpropane (DOTAP), increase endosomal escape and delivery of nanoparticle cargo into the cytosol.[36] Finally, more specific targeting of particular tissues and cell types can be achieved by attachment of biomolecular recognition elements to the nanoparticles, which bind to ligand molecules on the surface of the target cells. Antibodies,[37] nucleic acid aptamers,[38] carbohydrates,[37] and even some small molecules[39] like folic acid can all serve as recognition elements that increase the efficiency of nanoparticle delivery to specific tissues.

Therapeutic nanoparticles deliver both small molecule and macromolecule drugs. Frequently, toxic chemotherapeutic agents are encapsulated inside the nanoparticle, as with Doxil®.[40] Nucleic acids are among the most promising and extensively researched cargo for nanoparticle therapeutics, because they can treat disease through mechanisms inaccessible to small molecule drugs, and because without encapsulation or functionalization on nanoparticles, unmodified nucleic acid is unstable in serum and provokes a Toll-like receptor (TLR)-mediated innate immune response.[41,42] Nucleic acid-carrying nanoparticles can suppress target gene expression by siRNA-mediated mRNA cleavage and gene silencing,[43] stimulate the immune system through interaction with TLRs,[44] and deliver DNA[45] or mRNA[46] gene therapies to express a therapeutic protein inside target cells. As one example, mRNA for the *FUS1/TUSC2* tumor suppressor gene has been delivered in DOTAP/cholesterol liposomes for the treatment non-small cell lung cancer; this treatment is now in phase II clinical trials for the treatment of non-small cell lung cancer.[47,48] Finally, therapeutic proteins can be delivered as cargo inside nanoparticles;[49] however, inefficient endosomal escape has limited the translation of nanoparticles for intracellular protein delivery.[50]

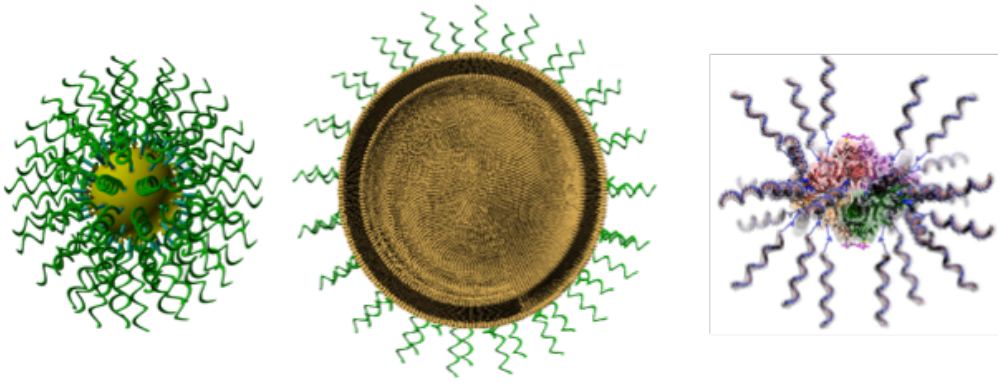


Figure 1. Structure of SNAs. Gold (left), liposome (center, cutaway), and protein (right) cores are radially functionalized with DNA oligonucleotides

1.4 Spherical Nucleic Acids

In 1996, a team led by Chad Mirkin attached a polyvalent shell of thiol-modified oligonucleotides to gold nanoparticles via a gold-thiolate bond, generating the first spherical nucleic acids (SNAs).[51] Both the core and the oligonucleotide composition can be varied (**Figure 1**). In addition to AuNPs, SNAs have been synthesized with cores made from, such diverse materials as iron oxide nanoparticles,[52] metal-organic

framework nanoparticles,[53] quantum dots,[54] silica nanoparticles,[55] pH-responsive block copolymer nanoparticles,[56] liposomes,[57] proteins,[58] and micelles.[59] There are even coreless spherical nucleic acids formed by crosslinking oligonucleotides on an AuNP and then dissolving the gold.[60] These empty SNAs helped confirm that it is the dense and highly oriented oligonucleotide shell that cause gives SNAs the unique properties that distinguish them from linear or circular nucleic acids. Oligonucleotides in an SNA bind to complementary nucleic acids with dissociation constants roughly two orders of magnitude higher than equivalent linear nucleic acids, and nucleic acid duplexes in an SNA not only have higher melting temperatures than linear nucleic acid duplexes; but also have much sharper, narrower, and more cooperative melting transitions as measured by the full width at half maximum of the melting curve's first derivative (~2ºC for SNAs versus >10º C for the same linear nucleic acid).[61] This tighter and more cooperative binding has the additional effect of destabilizing duplex mismatches, increasing the selectivity of oligonucleotide hybridization in SNAs relative to linear nucleic acids.[62] These properties stem primarily from the high concentration of cations permeating the highly anionic oligonucleotide monolayer; theoretical and experimental results suggest a loose network of cations each interact with and stabilize multiple oligonucleotides, such that perturbation of this shared ion cloud (for instance, due to dehybridization of a DNA duplex) destabilizes the surrounding dielectric environment (and with it, any surrounding nucleic acid duplexes).[63,64]

In 2006, it was discovered that SNAs enter mammalian cells rapidly and in large quantities (10^5-10^6 in an hour in cell lines).[65] As with SNAs' tight and cooperative hybridization to complementary DNA, this remarkable property is not shared by linear nucleic acids, and is a function the SNAs' 3-dimensional architecture. Subsequent research has uncovered that SNAs are actively taken up by mammalian cells expressing Scavenger Receptor A,[66] which tightly bind the dense and highly oriented oligonucleotide shell and endocytose the nanoparticles in a caveolae- and lipid raft-dependent process.[67] SNAs have now been tested on a multitude of mammalian cell lines and enter almost all of them, with the exception of red blood cells.[66]

Research in Mirkin lab has uncovered that, in addition to rapid uptake by mammalian cells, SNAs have several other remarkable biological properties that distinguish them from their linear counterparts. While linear nucleic acids are rapidly degraded by nucleases in serum and inside cells, SNAs resist nuclease degradation in these environments.[68] This nuclease resistance is hypothesized to result from the dense cloud of ions around SNAs, which inhibits the nuclease enzymes. And while most linear nucleic acids trigger a nonspecific TLR-mediated innate immune response in cells,[69] SNAs are non-immunogenic, provoking a 25-fold lower immune reactions than the same sequence of linear nucleic acid in macrophages.[70]

1.5 SNAs in Diagnostics

The unique properties of SNAs have been exploited to develop many assays for detecting analytes of interest. The first SNAs to be developed had AuNP cores, and the plasmonic properties of the AuNPs were used to construct colorimetric detectors of nucleic acids. These were solutions of two non-complementary SNAs which aggregated and changed color from red to blue in the presence of a specific polynucleotide sequence that could hybridize to and form a bridge between both SNA sequences.[71] Because of the cooperativity of SNA hybridization, this assay was highly specific, able to distinguish between sequences with single base mismatches. Subsequent SNA assays exploiting the same aggregation-triggered shift in AuNP solution color were developed to distinguish between different types of DNA-binding molecules.[72]

After colorimetric, aggregation-based assays, scanometric assays were the next SNA-based diagnostics to be developed.[73] In scanometric assays, a glass slide is functionalized with an oligonucleotide complementary to the target polynucleotide of interest, which hybridizes and captures the target from the sample solution. Then, the slide is incubated with AuNP-SNAs complementary to a different region of the target polynucleotide, forming a three-component sandwich with immobilized SNAs on the slide. After washing, the slide is incubated with a plasmonic metal (silver or gold) salt reduction solution, and the AuNPs catalyze the selective deposition of plasmonic metal onto the SNAs. These enlarged metal particles scatter light with extraordinary efficiency, and can be detected by imaging the slide in a laser scanner. The first SNA-based scanometric detection assays were capable of detecting oligonucleotide sequences with

two orders of magnitude greater sensitivity than an analogous fluorophore-based assay.[73] By functionalizing SNAs with antibodies, forming a sandwich with a target antigen and antibody-modified magnetic particles, and then hybridizing the captured SNAs on an oligonucleotide-functionalized chip, the scanometric assay was adapted to detect proteins at attomolar concentrations.[74] The scanometric assay has since demonstrated ultrasensitive detection of prostate serum antigen,[75] an important protein biomarker of prostate cancer. Moreover, by functionalizing an array of capture oligonucleotides onto a glass slide, the scanometric assay has been used to detect circulating microRNA (miRNA) patterns associated with prostate cancer in patient samples, and has been able to identify high-risk aggressive cancers based on patients' circulating miRNA profile.[76] Importantly, there is a trade-off between sensitivity and complexity in these SNA-based diagnostic assays. While the colorimetric SNA assays take place in a one-pot reaction and have a simple, visible readout, they are much less sensitive than the scanometric assays, which require multiple incubation steps and specialized equipment to complete. An approach to reconciling this tradeoff will be discussed in chapter 2.

After SNAs were discovered to rapidly enter mammalian cells, a third class of SNA-based diagnostic was developed: the NanoFlare.[77] NanoFlares are SNAs with AuNP cores functionalized with thiolated DNA oligonucleotides that are complementary to a target nucleic acid sequence of interest. A shorter, fluorophore-labeled DNA oligonucleotide called a flare strand is hybridized to the thiolated strands, with the fluorophore positioned close to the gold core. In the absence of the target sequence, the AuNP's surface plasmon resonance quenches the fluorescence of the flare. When the target sequence is present, it binds and hybridizes to the complementary thiolated DNA attached to the gold, displacing the flare strand. No longer quenched by close proximity to the gold, the flare's fluorophore fluoresces to generate a signal which enables the detection and quantification of the target sequence. NanoFlares are able to detect specific mRNA sequences in living cells without requiring a cationic transfection reagent. Importantly, their ability to do so implies that some fraction of SNAs are able to escape the endosome and enter the cytosol following cell uptake.

NanoFlares have subsequently been modified and improved in multiple ways. Multiplex NanoFlares, in which a single SNA can detect two different mRNA sequences, have enabled cell-by-cell internal controls in flare experiments and thereby improved quantitation of cancer-associated mRNA in living cells.[78] By measuring aberrantly high mRNA levels of metastasis-associated genes like vimentin, NanoFlares have been used to detect and isolate tiny numbers of tumor cells from human whole blood and mouse xenograft models.[79] Sticky-Flares, in which the fluorophore-labeled oligonucleotide rather than the AuNP-bound oligonucleotide hybridizes to the mRNA, enable the simultaneous quantification and tracking of mRNA expression and localization in live cells.[80] Finally, aptamer NanoFlares have been developed to detect the small molecule ATP both extracellularly and intracellularly.[81] Importantly, because this is the only published example of an aptamer NanoFlare detecting a small molecule, it is unclear how generalizable this diagnostic architecture is, and it is unknown how aptamer NanoFlare behavior and design rules may differ from either linear NanoFlares or other aptamer-based biosensing architectures. These questions will be explored further in chapter 3.

1.6 SNAs in Therapeutics

The combination of high cell uptake, low immunogenicity, nuclease resistance and tight oligonucleotide binding makes SNAs promising candidates for a plethora of therapeutic applications, including knockdown of gene expression, immune modulation, and protein delivery. Because SNAs are densely functionalized with oligonucleotides, the first therapeutic avenue to be explored was oligonucleotide-mediated gene regulation. The first paper on cellular SNA uptake demonstrated that the DNA oligonucleotides in the SNA shell, could reduce the expression of targeted genes with complementary mRNA sequences.[65]

After the early antisense SNA experiments, gene knockdown efficacy was increased by replacing the single-stranded antisense DNA oligonucleotides with double-stranded silencing RNA (siRNA).[82] Upon entry into the cytosol, the siRNA sequences are cleaved off the particle by Dcr-2 and loaded into eukaryotic Argonaute nucleases, which then chop up any mRNA containing a complementary sequence to the siRNA.[83]

The first siRNA-SNA conjugates consisted of a gold nanoparticle attached via gold-thiol bond to a double stranded RNA duplexes with one thiolated strand, and backfilled with thiolated oligo-ethylene glycol.[82] Like other SNAs, these constructs are nontoxic, non-immunogenic, resist nuclease degradation compared to linear RNA duplexes with the same sequence, and rapidly enter mammalian cells. These siRNA-SNA conjugates knock down targeted gene expression *in vitro* without the need for transfection reagents. In a mouse model of the deadly brain cancer glioblastoma multiforme, siRNA-SNAs were able to cross the compromised blood-brain barrier and knock down *Bcl2L12* oncogene expression, reducing tumor burden.[84] siRNA-SNAs have also been shown to knock down expression of the insulin-resistance mediating gene GM3 in human skin samples, and to improve wound healing in a mouse model of diabetes.[85] siRNA-SNAs have also been constructed with a liposomal core.[57] In these constructs, the gold nanoparticle and thiolated RNA are replaced with a 30 nm liposome and tocopherol- or cholesterol-modified RNA duplexes, and the hydrophobic tocopherol or cholesterol moiety intercalates into the liposome to form the SNA structure. Liposomal siRNA-SNAs are a biodegradable SNA-based plaform for siRNA delivery, and also knock down gene expression *in vitro*. Importantly, as with NanoFlares, the efficacy of gene regulating SNAs is further evidence that SNAs are capable of endosomal escape, because these SNAs' mechanism of action depends on interacting with cytosolic mRNA. However, the efficiency of endosomal escape is small enough that it is difficult to accurately measure.[86]

In addition to gene regulation, SNAs have been developed as delivery vehicles for small molecule therapeutics. The chemotherapeutic drug cisplatin binds and intercalates between DNA bases.[87] Incubating SNAs with with cisplatin enabled loading of the chemotherapeutic into the SNAs' oligonucleotide shell, followed by delivery of the drug upon cellular uptake of SNAs. SNA-mediated delivery increased the cytotoxicity of cisplatin in several cell lines.[88] Similarly, the chemotherapeutic paclitaxel has been conjugated to SNAs and delivered into cells; this SNA-mediated delivery increased the solubility of paclitaxel by 50-fold.[89]

Some of the most promising therapeutic avenues for SNAs involve modulation of the immune system. Although most SNAs provoke a minimal immune response, Mirkin lab has developed SNAs with specific sequences that can stimulate or suppress an innate immune response.[90] Immunostimulatory SNAs (IS-SNAs) consist of a spherical nanoparticle core functionalized with a DNA sequence known to bind and stimulate Tol-Like Receptor 9 (TLR9) in the endosome. IS-SNAs can have a gold core and thiol-modified oligonucleotides, or a liposome core with tocopherol-, cholesterol- or phospholipid-modified DNA with a phosphodiester or phosphorothioate backbone.[91] IS-SNAs efficiently enter the endosome and provoke an inflammatory cytokine immune response hundreds of times higher than linear nucleic acids with the same sequence, both *in vitro* and *in vivo*.[90] Moreover, IS-SNAs can be further functionalized by conjugating antigenic peptides to DNA oligonucleotides and hybridizing the peptide/DNA conjugates to the strands of the SNA. These rationally designed antigenic IS-SNAs stimulate a humoral (antibody-mediated) immune response to the antigen. Antigenic IS-SNAs can stimulate the immune system to specifically attack tumor cells expressing the antigen, thereby reducing tumor volume and increasing survival time in mouse models.[90, 92]

Just as some nucleic acid sequences stimulate TLR9 signaling, others suppress it. Immunoregulatory SNAs (IR-SNAs) consist of a nanoparticle functionalized with a DNA sequence known to suppress TLR9 signaling. IR-SNAs suppress macrophage innate immune responses *in vitro*, and decrease liver fibrosis in a mouse model of the inflammatory liver disease nonalcoholic steatohepatitis.[93] Immunomodulatory SNAs thus show promise as both treatments for cancer and inflammatory disease.

While most applications of spherical nucleic acids have thus far focused on the delivery of DNA or RNA into cells, recent work has explored the delivery of peptides and protein. As discussed above, antigenic peptides have been conjugated to oligonucleotides and hybridized to IS-SNAs as cancer vaccines. Peptide antigens have also been encapsulated in liposomal IS-SNAs and directly conjugated to cholesterol-modified, liposome-intercalating immunostimulatory oligonucleotides on the SNA surface, though these architectures were less effective than the hybridized antigen at provoking an immune response to antigen-

expressing tumors in mouse models.[92] The first full-length proteins attached to SNAs were antibodies, functionalized with ~2 N-hydroxysuccinimide-tetraethylene glycol-azide (NHS-PEG_4-N_3) moieties that covalently bound lysine primary amines.[94] Alkyne-modified DNA strands were conjugated to the azide-modified antibodies via click chemistry, and then the DNA-modified antibodies were hybridized to gold SNAs and used to selectively target and knock gene expression down in certain cancer cell types. The early immunomodulatory SNA work followed this model, functionalizing the model protein antigen ovalbumin with a small number of DNA strands and hybridizing the protein to the surface of an SNA. Another SNA-mediated protein delivery approach is to place the protein at the core of the SNA. This has been achieved by functionalizing most of the lysines on a protein with NHS-PEG_4-N_3 and then clicking DNA on, generating a protein-core SNA (proSNA) with a protein core.[58] These proSNAs have been shown to deliver functional enzymes, in particular ß-galactosidase, into cells *in vitro*.[95] These studies all demonstrate that SNAs could be an effective platform for delivering proteins into mammalian cells, and provide evidence that some fraction of SNAs are able to escape the endosome. However, the mechanism enabling SNA endosomal escape remains unknown, and the difficulty of quantifying endosomal escape have meant that it remains murky how SNA parameters might be tuned to increase endosomal escape. Moreover, while SNA-protein conjugates have been developed and demonstrated to enter cells, no study thus far has proven that protein delivered by SNA can enter the cytosol, because measurements of protein delivery to date do not differentiate between endosomal protein and cytosolic protein. These questions will be explored further in chapter 4.

CHAPTER 2: Dual Readout Assay for Device-Free and Sensitive Anthrax Biomarker Detection.

Portions of this chapter are adapted from Larkin et al., *Anal. Chem.*, **2020**

Collaborators include Chad Mirkin, Hang Xing, Vis Garimella, Gokay Yamankurt, & Alexander Scott

2.1 Introduction

Technologies for the detection of disease biomarkers are key to improving both healthcare and biosecurity around the world.[96] For example, anthrax is a severe bacterial infection caused by handling animal products or other materials contaminated with Bacillus anthracis spores. Pathogenic exposure to B. anthracis results in up to 89% mortality without proper treatment, and an estimated 2,000 anthrax cases are reported annually.[97] Anthrax pathogenesis depends on an exotoxin consisting of three protein components: protective antigen (PA), lethal factor (LF), and edema factor (EF). The 83 kDa PA protein (PA_{83}) is cleaved by protease to yield a 63 kDa fragment (PA_{63}), which self-assembles into heptamer and octamer rings.[98] The PA heptamer subsequently binds with LF and EF, is endocytosed, and forms a pore that translocates LF and EF from the endosome to the cytosol.[99] Because of PA_{83}'s role as an exotoxin protein expressed early during anthrax infections, and because PA_{83} levels track levels of bacteremia in anthrax animal models,[100] PA_{83} has been used for years as a biomarker for the early detection of anthrax.[101, 102, 103, 104, 105, 106] Methods to identify and develop bioassays specific for PA and other such disease biomarkers are of significant value.

Immunoassays that use antibodies as target recognition elements are the most widely used methods for biomarker detection because of their speed, ease of use, and capacity to detect a wide range of biomarkers and biomolecules.[107] Conventional immunoassays conjugate antibodies to fluorophores or enzymes to convert target binding to detectable fluorescent or colorimetric signals.[108, 109] However, these enzymatic fluorogenic and chromogenic methods have well-known drawbacks, including low stability, pH and temperature sensitivity, and limited sensitivity.[110, 111]

Over the past twenty years, nanomaterials with tailorable physical properties have been employed in biomarker assays that compare favorably with the molecular fluorophore or enzyme methods on sensitivity.[112, 113, 114, 115] A variety of nanoparticle-based readouts, including colorimetric,[116, 117, 118, 62, 71, 119] fluorescent,[78, 79, 81, 104, 120, 121, 122] light scattering,[75, 76, 123] electrochemical,[124, 125] and Raman scattering,[126, 127] show promise for the development of high sensitivity detection systems. However, a general tradeoff is observed between high assay sensitivity and high sample throughput.

For example, anisotropic platinum nanoparticles (PtNPs) and Pt-coated gold nanoparticles (AuNPs) have been deployed in assays as robust, enzyme-free replacements for horseradish peroxidase, where Pt catalyzes the decomposition of H_2O_2 and oxidation of a chromogenic substrate to produce a colorimetric signal.[119, 128, 129, 130] Such assays require only a few hours of processing time, can analyze many parallel (96-384) samples, and enable device-free visual detection of the target that, in principle, can function in point-of-care or field tests; but their limit of detection is typically confined to the nanomolar to picomolar range.[26, 130]

By contrast, scanometric AuNP-based assays have achieved ultrasensitive detection of protein and nucleic acid targets by sandwiching the target between two recognition elements, one immobilized on a glass slide and one attached to the AuNP.[74, 75, 76, 123, 131] By reducing Ag^+ or $AuCl_4^-$ ions from solution onto the AuNPs, the light scattering signal in a laser scanning instrument can be amplified to achieve detection of femtomolar to attomolar concentrations of target molecules. However, such assays typically require longer processing time and a specialized scanning instrument; and while the glass slides can accommodate multiplexed analysis of the biomarkers in each sample, the number of samples that can be analyzed in parallel is limited.

The tradeoffs between assay field deployability, sample throughput, and assay sensitivity can be reconciled with dual-readout nanoparticle assays, which generate two different types of signal from the same constructs. By combining orthogonal detection methods with different sensitivities, dual readout assays have been shown to lower the limits of detection and quantitation,[132] expand the dynamic range,[21] and enable both high-throughput and ultrasensitive target detection.[131, 133]

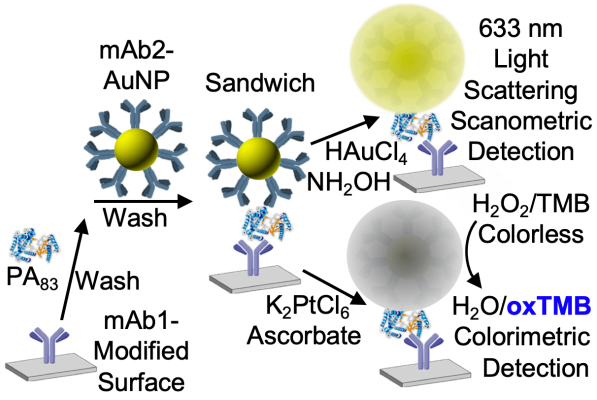


Figure 2. Dual-readout AuNP-based immuno-assay to detect anthrax protective antigen.

We present a dual-readout, colorimetric and scanometric sandwich immunoassay by depositing either Pt or Au onto antibody-AuNP conjugates (**Figure 2**). The higher-throughput Pt-based colorimetric readout was used to screen for monoclonal antibody sandwich pairs that bind to anthrax protective antigen (PA_{83}), detecting nanomolar concentrations of PA_{83} in both PBS and human serum. The Au-based scanometric readout showed a 1000-fold increase in assay sensitivity with the same nanoparticles, enabling detection of sub-picomolar PA_{83} concentrations.

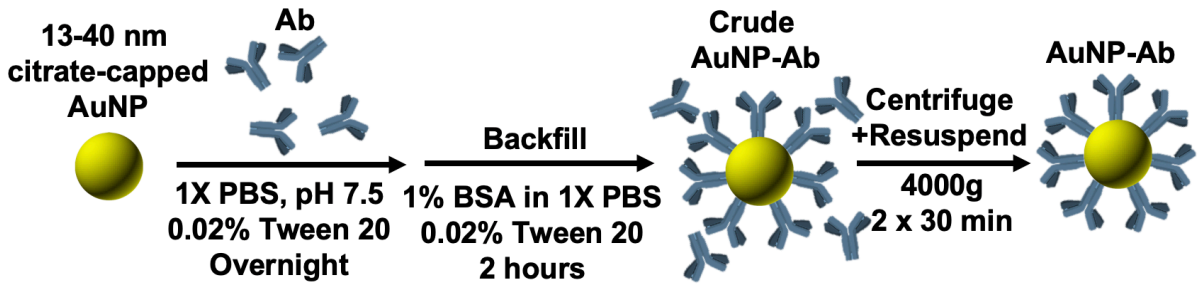


Figure 3. AuNP-Ab synthesis workflow. Abs adsorb to citrate capped AuNPs, which are then backfilled with BSA. Centrifugation removes unbound antibodies.

2.2 Design of Assay

The dual-readout sandwich immunoassay begins with the immobilization of one set of antibodies (Ab1) via lysine conjugation onto an *N*-hydroxy-succinimidyl-ester (NHS)-modified surface (either in a 96-well plate for colorimetric detection, or on a glass slide for scanometric detection). In parallel, a facile synthesis of Ab-AuNP conjugates is performed by mixing and incubating a second set of antibodies (Ab2) with AuNPs in a buffered solution. Strong Ab2 adsorption to AuNPs proceeds through electrostatic, hydrophobic, and cysteine-gold interactions,[9] after which the AuNP-Ab2 conjugates are blocked by incubation in a bovine serum albumin (BSA) solution and cleaned via centrifugation (**Figure 3**), generating monodisperse nanoparticles that can bind selectively to the antibody's antigen (**Figure 4**). The Ab1-modified surface is incubated with samples to capture the target molecule and washed with BSA solution to block nonspecific binding. The solution of Ab2-AuNP conjugates is then incubated on the surface, generating an Ab1-PA_{83}-Ab2-AuNP sandwich structure. Colorimetric detection with signal amplification is

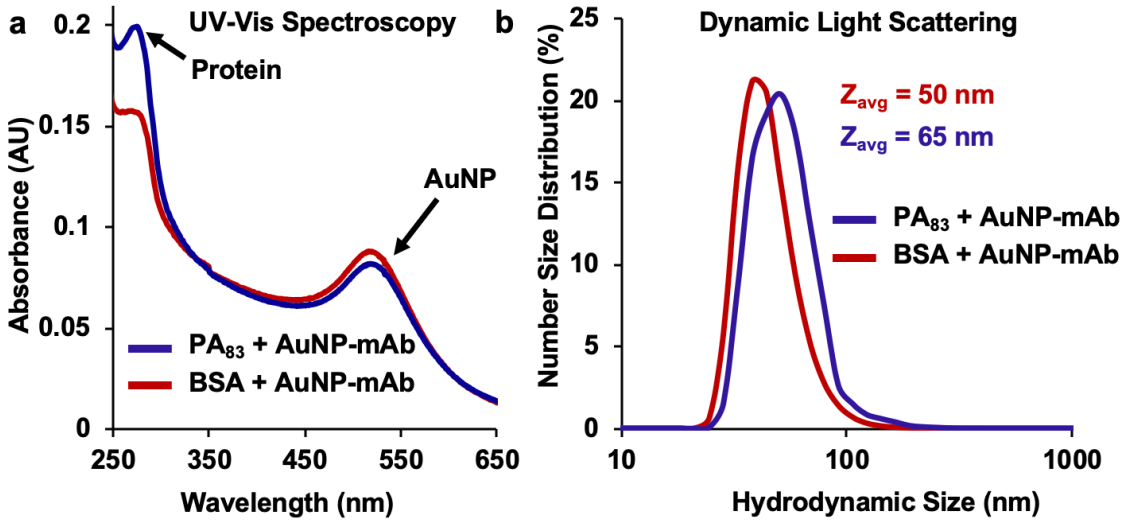


Figure 4. Characterization of Ab-AuNP conjugates after 1 week refrigeration at 4ºC. (a) UV-vis absorption and (b) DLS results show that AuNPs-Abs are stable (no blue-shift in the AuNP absorption peak) and selectively bind to PA_{83}.

achieved through incubation with a reducing agent (ascorbic acid) in a platinum salt solution to selectively reduce Pt onto the nanoparticles. This process forms a Pt shell on the AuNPs which catalyzes the splitting of hydrogen peroxide and subsequent oxidation of the chromogenic dye 3,3',5,5'-tetramethylbenzidine (TMB) to yield a blue color which can be quantified on a plate reader.[130] To achieve scanometric detection, the Ab1-PA_{83}-Ab2-AuNP sandwich is generated on an NHS-activated glass slide and incubated in a solution of Au^{3+} salt with a reducing agent (hydroxylamine) to selectively reduce gold onto the nanoparticles. After washing, light scattering from the gold on the slide is quantified in a Scano-miR instrument.

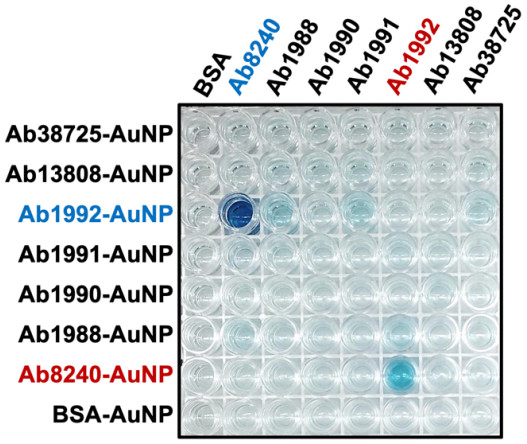


Figure 5. Pairwise anti-PA_{83} antibody screen for binding and detection of PA_{83} in a sandwich assay. Ab1992 and Ab8240 form a sandwich and detect PA_{83}. Each column contains a different Ab immobilized in the well, while each row is incubated with a different mAb-AuNP nanoparticle. Ab1992 and Ab8240 form a sandwich pair and detect PA_{83} in both orientations (blue and red).

2.3 Screen for Anti-PA_{83} Monoclonal Antibody Sandwich

To evaluate the scalability of the assay's colorimetric readout method, we screened monoclonal antibodies (mAbs) of anthrax protective antigen to discover pairs that could function in a sandwich assay. While many

anti-PA_{83} monoclonal antibodies are commercially available and a few anti-PA_{83} monoclonal antibody sandwich pairs have been reported,[134] no sandwich pairs of monoclonal antibodies for PA_{83} are commercially available, potentially limiting the long-term reproducibility of any given PA_{83} immunoassay.[135, 136]

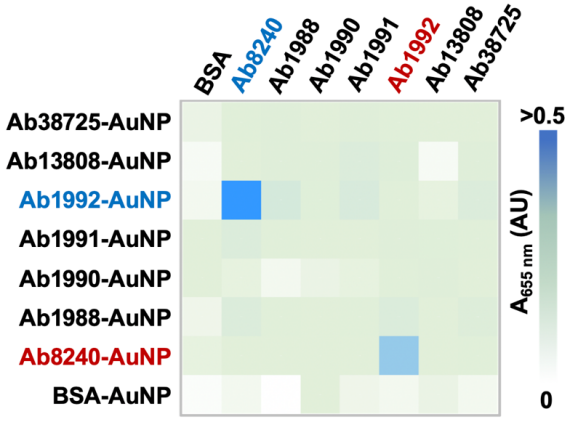


Figure 6. Heatmap of pairwise screen of seven commercial anti-PA_{83} antibodies for binding and detection of PA_{83} in a sandwich assay.

We addressed this by investigating 7 different anti-PA_{83} antibodies from Abcam: Ab8240, Ab1988, Ab1990, Ab1992, Ab13808 and Ab38725 (**Figure 5**). The seven PA_{83}-binding antibodies were immobilized in 8-well rows to form an 8x8 well array on an NHS-modified 96-well plate and washed with a BSA blocking solution. All wells were incubated with 500 nM PA_{83}, washed with blocking solution, and then incubated pairwise with different mAb-AuNP conjugates to generate all possible mAb1-PA_{83}-mAb2-AuNP sandwiches. After washing again with blocking solution, a Pt salt solution was reduced onto the immobilized particles. After gently rinsing the wells with DI water, they were incubated with TMB and 50 mM H_2O_2. One mAb pair, Ab8240 and Ab1992, was visually identified as a potential hit, as it gave a colorimetric and UV-Vis absorbance signal 20-fold higher than BSA control and 8-fold higher than any other antibody pair (**Figure 5 and 6**). Although the initial screen indicated that immobilized

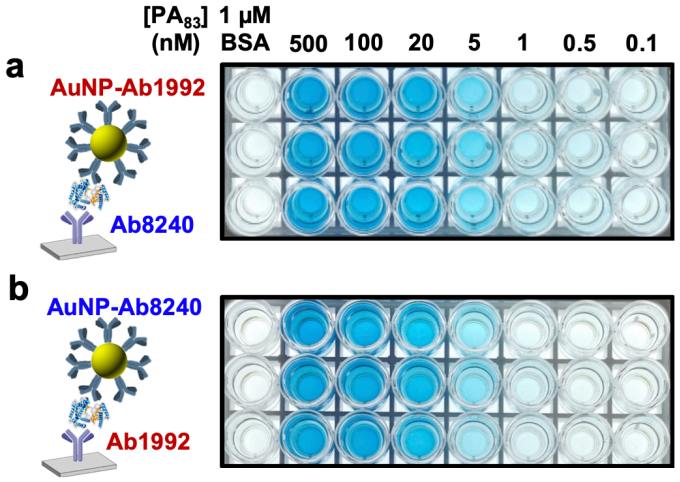


Figure 7. Sandwich assay performs device-free PA_{83} detection in either antibody orientation. (a) Detection of PA_{83} in PBS with immobilized Ab1992 and Ab8240-AuNP. (b) Detection of PA_{83} in PBS with immobilized Ab8240 and Ab1992-AuNP.

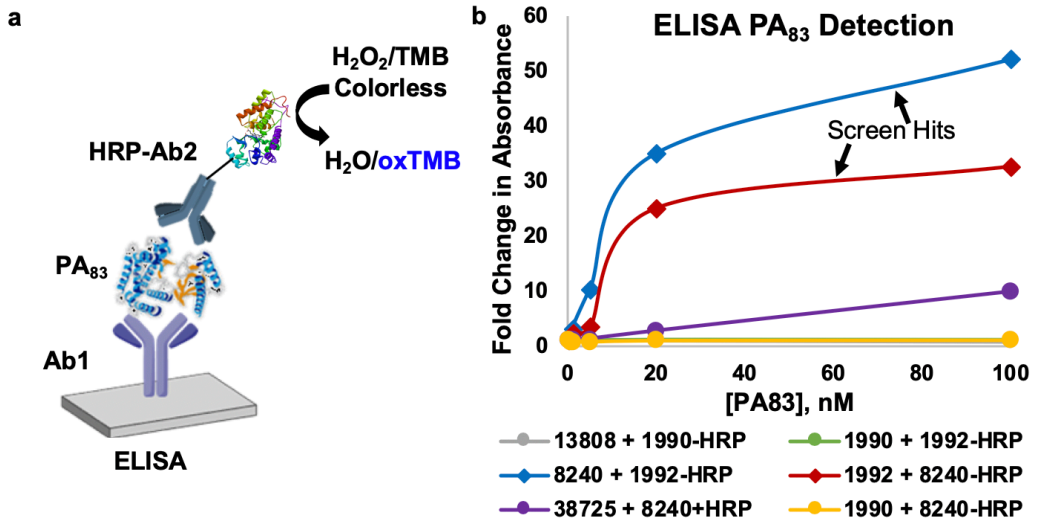


Figure 8. Screened antibody pair functions in an ELISA. (a) Scheme of ELISA sandwich detection strategy for PA_{83}, incorporating an immobilized antibody and a horseradish peroxidase-conjugated antibody. (b) ELISA detection curve of PA_{83} in 1X PBS, with different PA_{83}-binding antibody pairs, 15 min after incubation with 50 mM H_2O_2 in TMB.

Ab8240 and Ab1992-AuNP generated greater colorimetric signal than immobilized Ab1992 and Ab8240-AuNP, further experiments showed that both antibody pair orientations in the sandwich gave similar detection sensitivity (**Figure 7**). To validate the obtained antibody pairs, we tested them in a standard

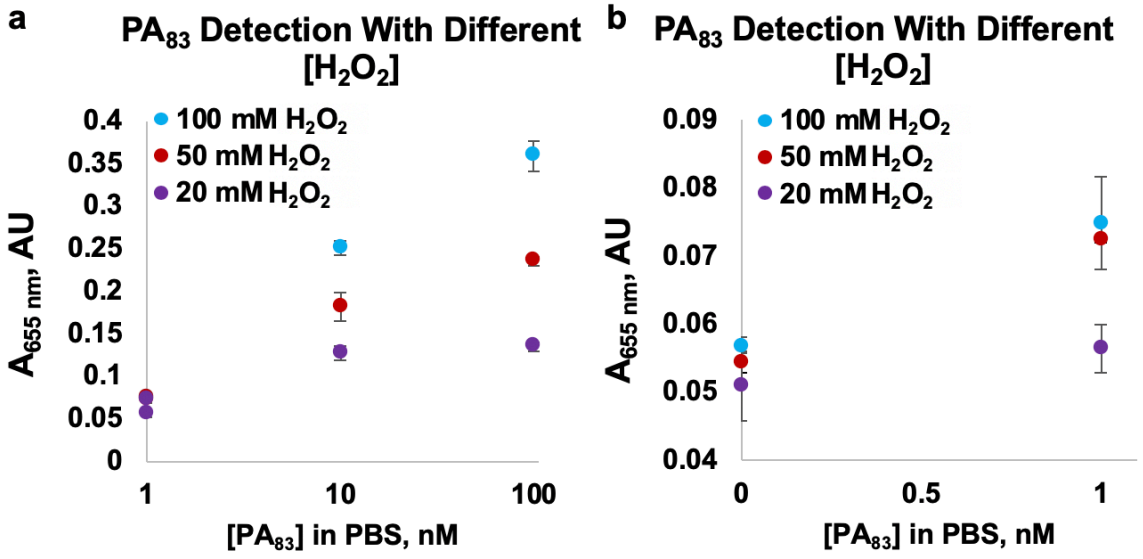


Figure 9. Optimization of sandwich assay for H_2O_2 concentration. (a) Sandwich assay response to 1, 10, and 100 nM PA_{83} in 1X PBS, using TMB blended with 20, 50, or 100 mM H_2O_2. (b) Triplicate sandwich assay response to 0 and 1 nM PA_{83}, using TMB blended with 20, 50, or 100 mM H_2O_2.

ELISA. #Ab1992 and #Ab8240 readily detected PA_{83} in an ELISA, while other antibody pairs either displayed no response above background or only responded to higher PA_{83} concentrations (**Figure 8**).

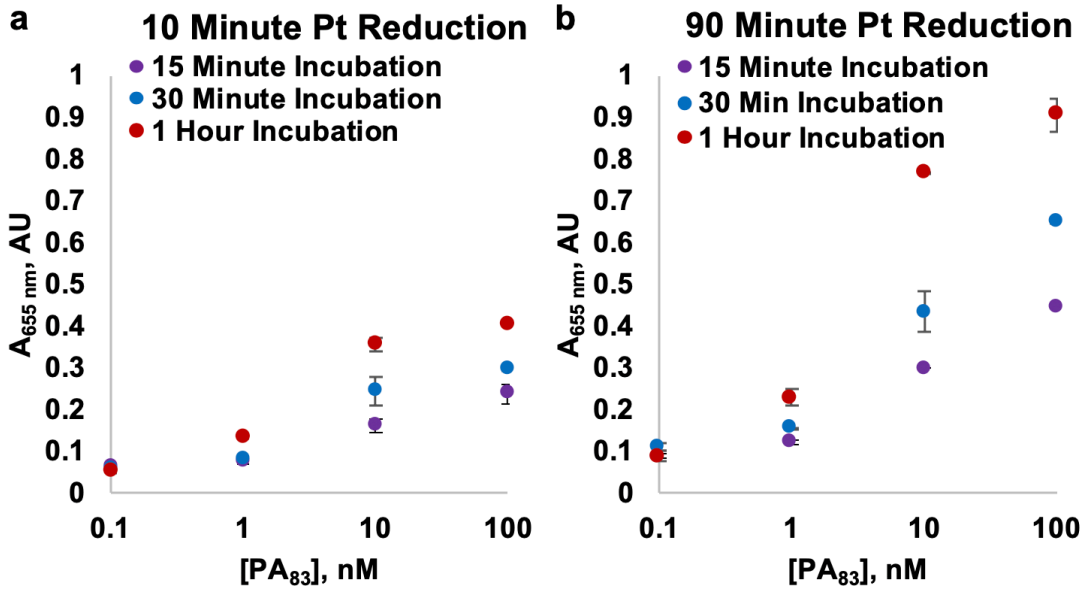


Figure 10. Optimizing time of sandwich assay incubation steps. (a) PA_{83} detection curves with 15 min, 30 min, or 1 h incubation steps and a 10 min platinum reduction. (b) PA_{83} detection curves with 15 min, 30 min, or 1 h incubation steps and a 90 min platinum reduction.

2.4 Colorimetric PA_{83} Detection in Phosphate-Buffered Saline

Having discovered an antibody pair with high selectivity for PA_{83}, we explored its ability to colorimetrically detect PA_{83} in PBS. Reaction conditions including H_2O_2 concentration (20-100 mM), Pt reduction time (10-120 min), and PA_{83} and mAb-AuNP incubation times (15-60 min) were varied to maximize colorimetric response to PA_{83} and minimize background signal (**Figures 9-11**). 1 h PA_{83} and mAb-AuNP incubation steps, a 120 min Pt reduction step, and 100 mM H_2O_2

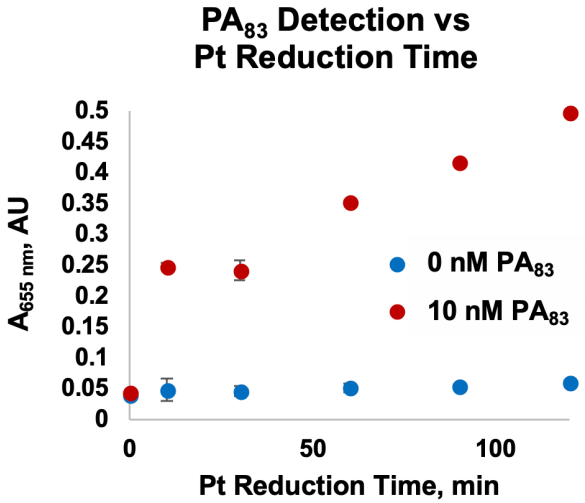


Figure 11. Optimization of sandwich assay for platinum reduction time. Sandwich assay response to 0 or 10 nM PA_{83} in PBS, after running the platinum reduction for a range of reaction times.

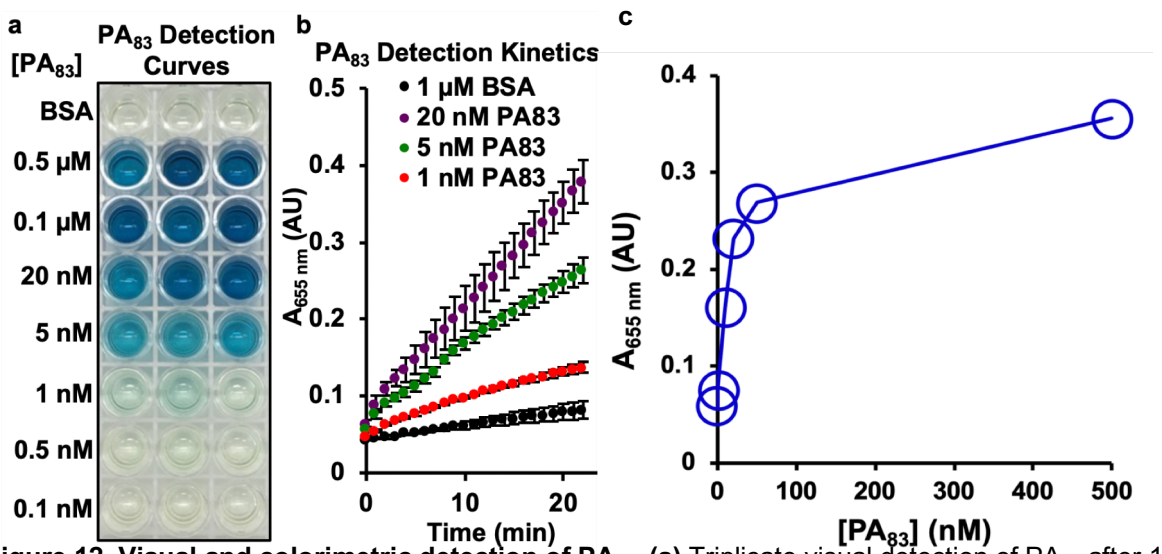


Figure 12. Visual and colorimetric detection of PA_{83}. **(a)** Triplicate visual detection of PA_{83} after 1 h Pt reduction and 20 min of TMB/H_2O_2 reaction. Control condition: 1 μM BSA. **(b)** Kinetics of PA_{83} detection with TMB/ H_2O_2. **(c)** Colorimetric PA_{83} detection calibration curve. Concentration-dependent calibration curve for Pt-based colorimetric detection of PA_{83} in 1X PBS with the absorbance at 655 nm measured on Synergy H4 Plate Reader.

concentrations in the TMB/H_2O_2 solution produced the largest response to the lowest [PA_{83}] relative to background signal. The colorimetric readout provides the ability to detect PA_{83} over the 1 to 500 nM dynamic range (**Figure 12**), with visual detection of 1 nM PA_{83} after a 20 min incubation and colorimetric detection of 1 nM PA_{83} within 2 min of adding TMB in a plate reader.

2.5 PA_{83} Detection in Human Serum

One of the primary advantages of sandwich assays is their ability to specifically detect analytes in a complex solution, like bodily fluids. Therefore, the ability of the antibody pair to detect PA_{83} in human serum was tested over the 0.5 to 100 nM range in 1:1 PBS:human serum samples (**Figure 13**).

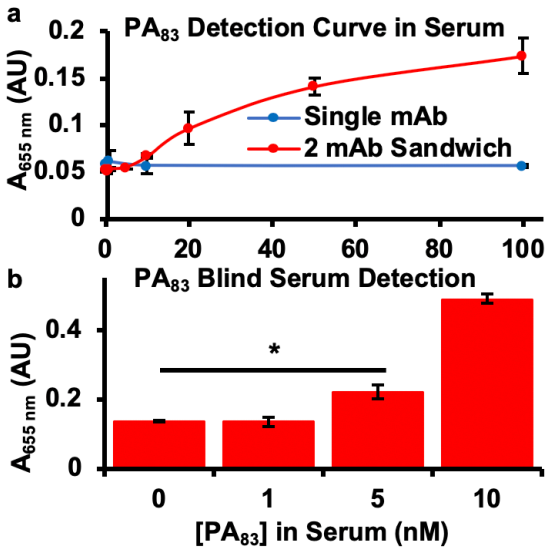


Figure 13. Detection of PA_{83} in serum. (a) Detection of PA_{83} spiked into 1:1 PBS:human serum with a single Ab (blue) and with the sandwich mAb pair (red). (b) Blind detection of nanomolar PA_{83} in human serum. (*p < 0.01, Student's t-test).

To determine the utility of the antibody sandwich, direct detection with a single mAb was attempted, by

incubating PA_{83}-spiked serum samples directly in unblocked NHS-ester modified wells, before adding the mAb-AuNP conjugate. After a 20 min incubation in colorimetric detection solution, the mAb sandwich displayed a dose response from 10 to 100 nM PA_{83}, while the single mAb failed to detect any PA_{83} concentration, thereby demonstrating the importance of the sandwich assay for detecting protective antigen in complex, physiologically relevant solutions. In a blinded experiment, colorimetric PA_{83} detection in 1:1 PBS:serum could be further improved to 5 nM PA_{83} by allowing the final incubation in colorimetric detection solution to run overnight. Because the serum is diluted 1:1 in PBS, these experiments demonstrate detection down to 10-20 nM PA_{83} (830-1660 ng/mL) in whole serum. This is within the physiological range of PA_{83} concentrations observed in the serum of rabbits (1-100,000 ng/mL) and guinea pigs (1-5,000 ng/mL) during the progression of inhalational anthrax.[136] However, the PA_{83} concentrations detected by this colorimetric assay correspond to more advanced stages of anthrax rather than the early, potentially treatable stage, at least in these two animal models; so more sensitive methods of signal amplification and detection are desirable to make an assay that could potentially enable early diagnosis and successful treatment of anthrax.

2.6 Scanometric Detection of PA_{83}

Although the colorimetric mAb-AuNP sandwich was successfully used to screen for and discover an antibody sandwich pair that could detect pathogenically relevant concentrations of PA_{83}, sensitivity to even lower concentrations could potentially enable earlier diagnosis and successful treatment. This is particularly important for anthrax, as the expression of protective antigen facilitates the endocytosis of the

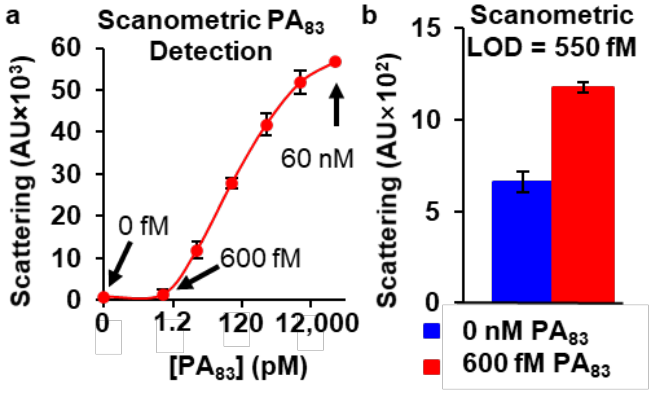


Figure 14. Scanometric detection of PA_{83} via gold reduction onto mAb-AuNP sandwich. (a) Quantification of scanometric PA_{83} detection using 633 nm light scattering. (b) Detection of sub-picomolar concentrations of PA_{83}.

lethal factor and edema factor toxins required for disease progression.[98, 99] We therefore sought to determine whether measuring the scanometric readout of the sandwich immunoassay increased detection sensitivity (**Figure 14**). mAb 1992 was functionalized on an NHS-ester activated glass slide and incubated first with PA_{83} and then with mAb-AuNP 8240, with blocking steps in between. A gold reduction solution was added to the slide to amplify the gold signal. Scattering of 633 nm laser light across the slide was collected in a Scano-miR instrument (**Figure 15**) and quantified with GenePix software. The scanometric assay detected PA_{83} at concentrations ranging from 600 fM to 60 nM in PBS with 1% BSA and 0.02% Tween, with a limit of detection of 550 fM. This is over 1000 times more sensitive than the Pt-based colorimetric assay of PA_{83} in the same solution. These results underscore the observation that gold reduction and scanometric readout is a general strategy for increasing the sensitivity of antibody sandwich assays. This could be particularly useful for biomarkers of infection such as PA_{83}, for which early identification of the pathogen can be critical for successfully treating the disease.[97]

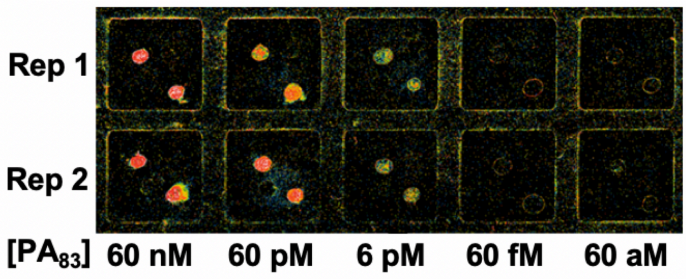


Figure 15. Representative light scattering image of an 8-well glass slide showing scanometric detection of PA_{83}.

2.7 Materials and Methods

2.7.1 Reagents. Citrate capped gold nanoparticles (13 nm and 40 nm) were purchased from Ted Pella or synthesized as previously described.[137] The seven screened Anti-PA_{83} antibodies (Ab8240, Ab1988, Ab1990, Ab1991, Ab1992, Ab13808, and Ab38725) were purchased from AbCam. *N*-hydroxy succinimide (NHS)-activated 96-well plates (divided into 8-well strips), NHS-activated glass slides for scanometric detection, aliquots of mAb 1992, and EZ-Link Plus activated peroxidase kits for horseradish peroxidase-antibody conjugation were purchased from Thermo Fisher. All other materials, buffers and reagents, including platinum and gold salts, were purchased from Sigma Aldrich.

2.7.2 Buffered solutions. Blocking solution, used for all blocking and washing steps during mAb-AuNP synthesis and PA_{83} detection assays, contained 1X phosphate-buffered saline (PBS), 1% bovine serum albumin (BSA), and 0.02% Tween 20. Platinum deposition solution, used during colorimetric detection, contained 10 mM citrate buffer (pH 3), 20 mM L-ascorbic acid, and 2 mM potassium hexachloroplatinate (K_2PtCl_6). Colorimetric detection solution contained 50 mM H_2O_2 in 3,3',5,5'-tetra-methylbenzidine (TMB); and was prepared fresh before each colorimetric detection experiment. Gold reduction solution contained 10 mM chloroauric acid ($HAuCl_4$) with 10 mM hydroxylamine (NH_2OH), and was prepared fresh before each gold reduction reaction.

2.7.3 Synthesis of Antibody-Coated Gold Nanoparticles. Antibodies were noncovalently adsorbed onto the surface of AuNPs, similar to previously described methods.[9] In a 15 or 50 mL plastic conical tube, a solution of 10 nM of citrate-capped AuNPs in water (40 nm diameter for dynamic light scattering (DLS) particle size measurements; 13 nm diameter for all other experiments) was adjusted to pH 7 using 0.2 M NaOH, and Tween 20 was added to a final concentration of 0.02%. Unmodified antibodies were added to a final concentration of 15 μg/mL, and the solution was mixed gently by inverting the tube 4-6 times. The tube was incubated overnight at room temperature in the dark. The mAb-AuNP mixture was then diluted 1:1 with blocking solution, gently mixed by inverting the tube 4-6 times, sealed by capping the tube and wrapping the cap in parafilm, and incubated for 3 h in the dark at room temperature. The crude mAb-AuNPs were cleaned via two rounds of pelleting (4,000 RCF for 30 min at room temperature in 1.5 mL low-retention plastic tubes), supernatant removal, and resuspension in blocking solution. This protocol generates antibody-mAb nanoparticles that maintain function for at least 3 months at 4 ºC.

2.7.4 *Characterization of AuNP-mAbs*. Nanoparticle concentration was measured by UV-Vis spectroscopy on a Cary5000 spectrophotometer (gold absorption peak at 520 nm, extinction coefficient $2.7*10^{-8}$ L/mol/cm). To characterize protein adsorption to the gold surface, particles were twice centrifuged at 4000 RCF for 30 min and resuspended in 1X PBS with 0.02% Tween (no BSA), then absorption at 280 nm was measured via UV-Vis. To measure the size of mAb-coated AuNPs, mAb-AuNPs were twice centrifuged at

4000 RCF and resuspended in 1X PBS (no BSA or Tween), diluted to 0.5-1 nM AuNP, and measured in a Malvern Zetasizer DLS instrument.

2.7.5 Making Antibody-Coated Wells. 50 μL of 2 mg/mL antibody solution was diluted with an equal volume of 1X PBS (pH 7.5) to make 100 μL of 1 mg/mL antibody. 10 μL of 1 mg/mL antibody solution was carefully and evenly added to the center of each well in a strip of 8 NHS ester modified wells. The strip was incubated overnight in a sealed chamber with a water reservoir (roughly 50% humidity). Then 100 μL of blocking solution was pipetted into to each well and gently mixed. After 2 min of incubation, the liquid was removed from the strip. These washing steps were repeated twice more. The strips were stored in a humid chamber before use to avoid them drying out. For screening, eight antibody-coated 8-well strips were made, one for each antibody and a BSA control. For further PA_{83} detection assays, mAb 1992 was functionalized on the NHS-activated surface.

2.7.6 Screening for Antibody Sandwich Pairs. 3 mL of 500 nM PA_{83} solution in 1X PBS and 1 mL of 1 μM BSA solution in 1X PBS were prepared. To each of the eight 8-well strips, 50 μL of the 500 nM PA_{83} solution was added. The strips were incubated in the humid chamber for 1 h, and then washed three times with blocking solution. Next, 50 μL of 10 nM of antibody functionalized nanoparticles was added to each well, such that every combination of immobilized antibody and mAb-AuNP was tested, along with the BSA negative controls. The strips were incubated in the humid chamber for 1 h and then washed with blocking solution three times. During incubation, a 4 mL batch of platinum deposition solution, which uses ascorbic acid to reduce Pt(IV) ions onto gold nanoparticles,[130] was prepared from stock solutions. 50 μL platinum deposition solution was added to each well. After incubating the strips for 1.5 h in the humid chamber, all wells were gently washed five times with DI water. 10 mL of colorimetric detection solution was prepared, and 100 μL was added to each well. Strips were incubated in a dark humid chamber for 20 min, avoiding light to minimize background signal. PA_{83} detection was assessed qualitatively by eye, imaged on an Alpha-Innotech FluorChem Q imager, and quantitatively measured via absorbance at 655 nm in a Bio-Tek H4 plate reader.

2.7.7 Colorimetric PA_{83} Detection Curves and Kinetics. Three 8-well strips were coated with mAb 1992, and 5 mL AuNPs coated with mAb 8240 were prepared as described above. A dilution series of PA_{83} in 1X PBS was prepared, comprising 1 mL each of 500, 100, 20, 5, 1, 0.5, and 0.1 nM PA_{83}, as well as a 1 μM BSA control. 50 μL of each PA_{83} concentration and the BSA control were added in triplicate to the coated 8-well strips. Incubation, washing, addition of mAb-AuNPs, and platinum reduction was performed as described above. Then a colorimetric detection solution was added to all the wells. The strips were immediately placed in a Bio-Tek H4 plate reader, and the absorbance at 655 nm was measured every minute for 25 min. The strips were then photographed with an Alpha Innotech FluorChem Q imager, under white light.

2.7.8 Serum PA_{83} Detection Curve. Triplicate mAb 1992-functionalized 8-well strips, and a 5 mL batch of 8240-functionalized AuNPs were prepared as described above. Human serum solution was prepared by diluting human serum 1:1 in 1X PBS. A dilution series of PA_{83} concentrations (500, 100, 20, 5, 2, 1, and 0.5 nM) was prepared by serially diluting 8 μM stock PA_{83} into human serum solution. Incubation of PA_{83} dilutions and the control 1 μM BSA in human serum, and subsequent colorimetric detection were performed as described above. The absorbance at 655 nm was quantified by plate reader after 20 min of incubation with colorimetric detection solution.

2.7.9 Scanometric PA_{83} Detection. 200 μL of 1 mg/mL mAb 1992 solution was prepared by diluting antibody stock 1:1 in 1X PBS. NHS-activated glass slides were divided into 10 wells by attachment of a rubber gasket. Two replicate 5 μL spots of mAb 1992 solution were pipetted into each well, and the slide was incubated overnight in the humid chamber. The slide wells were then washed three times with blocking solution. To determine the scanometric detection range of PA_{83}, a dilution series of PA_{83} concentrations (60 nM, 6 nM, 600 pM, 60 pM, 6 pM, and 600 fM) were prepared in 1X PBS, and incubated in two replicate wells for 1 h in the humid chamber. The wells were washed 3 times with blocking solution, and then incubated for 1 h with 10 nM of 8240-modified AuNPs. Wells were washed 3 more times with blocking solution, and then two gold reductions were performed to amplify the signal.[75]

To perform gold reductions, the slide was removed from the rubber gaskets. Solutions of 10 mM $HAuCl_4$ and 10 mM hydroxylamine (NH_2OH) were prepared in separate 10 mL syringes. The syringes were connected by a T-junction to a single output tube, to enable rapid mixing of the solutions. 4 mL of a 1:1 $HAuCl_4/NH_2OH$ mixture was spread evenly over the surface the slide for 30 s, then the slide was immersed in a DI water bath to wash gold reduction solution away, gently rinsed with running DI water, and blown dry with N_2. The gold reduction, washing, and drying steps were repeated once more. The slide was then imaged with a Tecan LS Reloaded scanner. Light scattering signal intensities from the two replicate spots for each PA_{83} concentration were quantified by GenePix Pro 6 software. The limit of detection was determined using 3.5 sigma above the negative control and fitting the data points to a Hill equation.

2.7.10 Monoclonal antibody orientation in PA_{83} sandwich. Three 8-well strips were coated with mAb 1992, and three were coated with mAb 8240. 5 mL each of gold nanoparticles (AuNPs) coated with either mAb 8240 or mAb 1992 were prepared as described above. Seven concentrations of PA_{83} (500, 100, 20, 5, 1, 0.5, and 0.1 nM) were prepared in 1X PBS as described in the main text. 50 μL of each PA_{83} concentration and the BSA control were added in triplicate to the coated 8-well strips. Incubations, washes, addition of AuNP-mAbs (AuNP-mAb 1992 to the 8240-coated wells, and AuNP-mAb 8240 to the 1992 coated wells), platinum reduction, and qualitative colorimetric detection with the Alpha-Innotech FluorChem Q imager was performed as described in the screening assay in the main methods section.

2.7.11 Optimization of H_2O_2 concentration in TMB/H_2O_2 solution. Three 8-well strips were coated with mAb 1992, and 5 mL AuNPs coated with mAb 8240 were prepared as described in the main text. Three concentrations of PA_{83}, 100, 10, and 1 nM, were prepared in 1X PBS. 50 μL of each PA_{83} concentration and the BSA control were added in triplicate to the coated 8-well strips. Incubations, washes, addition of AuNP-mAbs, and platinum reduction was performed as described in the screening assay in the main text. Three colorimetric detection solutions were prepared, by mixing 20, 50, or 100 mM H_2O_2 in TMB. The strips were then placed in a Bio-Tek H4 plate reader, and the absorbance at 655 nm was measured after 20 min of incubation.

2.7.12 Optimization of Pt reduction time. Five 8-well strips were coated with mAb 1992, and 5 mL AuNPs coated with mAb 8240 were prepared as described in the main text. 10 nM PA_{83} was prepared in 1X PBS as described in the main text. 50 μL of 10 nM PA_{83} was added to 18 wells, and 50 μL blocking solution was added to 18 control wells. 50 μL Pt reduction solution was added to each well. For both the 10 nM PA_{83} and the 0 nM control condition, 3 wells each were incubated with the Pt reduction solution for 0, 10, 30, 60, 90, and 120 min before rinsing 5 times with water. 100 μL colorimetric detection solution with 50 mM H_2O_2 in TMB was added to each well and the absorbance at 655 nm was measured in a Bio-Tek H4 plate reader after 20 min.

2.7.13 Optimization of incubation times. Six 8-well strips were coated with mAb 1992, and 5 mL AuNPs coated with mAb 8240 were prepared as described in the main text. Solutions of 100, 10, and 1 nM PA_{83} were prepared in 1X PBS as described above. 12 wells were filled with 50 μL of 100, 10, 1 nM PA_{83} or a blocking solution control. For each PA_{83} concentration, four wells were incubated with PA_{83} for 15 min, four were incubated for 30 min, and four were incubated for 60 min before washing 3X with blocking solution. 20 μL of 10 nM AuNP-mAb was added to each well, and incubated for the same amount of time as the PA_{83} solution (i.e. 15, 30, or 60 min depending on the well). Then, for each set of four replicate wells, two were incubated with Pt reduction solution for 10 min, and two for 90 min, before washing 5X with water. Colorimetric detection solution was added and absorbance after 20 min was measured as described above.

2.7.14 ELISA. 8-well strips functionalized with mAb 1992, 8240, 13808, and 38725 were prepared as described in the main text. 100 μL of 1 mg/mL mAb 1992, 8240, and 1990 were run through NAP-5 columns equilibrated in 1X PBS to remove the sodium azide preservative (required for the use of the HRP conjugation kit). Then horseradish peroxidase was conjugated to mAb 1992, 8240, and 1990 using an EZ-Link Plus activated peroxidase kit. A PA_{83} dilution series (100, 20, 5, 1, 0.5, and a 1 μM BSA control) was prepared, and incubated/washed in each 8-well strip as described above. A 1:100 dilution of each HRP-mAb was incubated in the wells for 1 h in the following arrangement: 1992-8240-HRP, 8240-1992-HRP,

13808-1990-HRP and 38725-8240-HRP sandwich pairs. The wells were then washed with blocking solution 3 times, and 100 μL colorimetric detection solution was added. After 20 min incubating in the dark, the wells were imaged in FluorChem Q imager and the absorbance at 655 nm was quantified on the plate reader. Fold change in absorbance was calculated and plotted by dividing the absorbance at each PA_{83} concentration by the absorbance of the negative control.

2.7.15 Blind Serum PA_{83} Detection. Two mAb-functionalized 8-well strips and a 5 mL batch of AuNP-mAbs were prepared as described above. Three replicate 50 μl aliquots of 10, 5, 1, and 0 nM PA_{83} in 1:1 serum:PBS were prepared and divided into three 50 μL aliquots in labeled 1.5 mL Eppendorf tubes. These aliquots were given to a third party, who randomly transferred their contents to fresh Eppendorf tubes which were numbered 1-12, recording in their notebook what concentration of PA_{83} was transferred to each numbered tube. The numbered tubes were returned to the experimenter without revealing their contents. The wells on the two mAb-functionalized strips were labeled 1-12, and 50 μL from the numbered aliquots were added to each corresponding well. The PA_{83} incubation, AuNP-mAb incubation, and Pt reduction were performed as described in the main text. Colorimetric detection solution was added and incubated overnight in a dark chamber, after which absorbance at 655 nm was measured in the plate reader. Finally, the key recording each numbered aliquot's PA_{83} concentration was retrieved from the third party.

CHAPTER 3: Modeling, Measuring and Screening Aptamer NanoFlares Toward Detection of Human Stress Biomarkers

---

Collaborators include Chad Mirkin, Caroline Kusmierz, Sasha Ebrahimi, Jorge Chavez, Peter Mirau, Nancy Kelly Loughnane, Rachel Krabacher, Monica Wolfe, and Alyssa Chinen

3.1 Introduction

Although some stress is required for optimal human function, too much stress is detrimental to performance and leads to the dysregulation of multiple physiological systems, damaging human health by impairing cardiovascular, immune, metabolic, and neuroendocrine function.[138, 139] Moreover, the same stressors can increase some individuals' performance, while overloading and damaging others' performance and health.[140] These realities have generated research interest into ways to measure individuals' stress levels, their allostatic load,[141] in order to better manage and improve performance, psychological and physiological wellbeing.

One approach to this challenge is to develop methods to quantify the concentration of biomarkers that are indicative of allostatic load in human samples such as sweat, saliva, urine and blood. Molecular biomarkers of neuroendocrine, cardiovascular, immune and metabolic health include cortisol, dehydroepiandrosterone-sulphate (DHEA-S), insulin, glycosylated hemoglobin, creatinine, albumin, pancreatic amylase, C-reactive protein, fibrinogen, serum triglycerides, serum cholesterol, high-density lipoprotein.[142] DHEA-S and cortisol, in particular, are widely studied as biomarkers of human stress. Both DHEA-S and cortisol are adrenal glucocorticoid steroid hormones, secreted as part of the hypothalamic-pituitary-adrenal axis of the fight-or-flight response.[143] However, while high cortisol concentrations induce physiological stress and suppress immune and digestive function,[144] high DHEA-S concentrations are correlated with lower levels of stress and improved neurophysiological health and performance.[145]

The standard methods for detecting and measuring DHEA-S and cortisol concentrations are liquid chromatography-tandem mass spectrometry (LC-MS/MS), and immunoassays like competitive ELISA.[146] While LC-MS/MS measures different glucocorticoid hormone levels with high accuracy, it requires an expensive specialty instrument and a high degree of technical expertise to run.[147] Immunoassays are simpler and less expensive to run than LC-MS/MS, but still require multiple time-consuming and potentially error-prone liquid handling steps, and suffer from cross-reactivity of the antibodies for structurally similar

glucocorticoid hormones, which reduces assay sensitivity and specificity.[148] A simple and specific one-pot assay for different glucocorticoid stress biomarkers would be valuable.

Oligonucleotide aptamers may provide a path to developing such an assay. These are single-stranded DNA or RNA oligonucleotides that adopt a 3-dimensional structure that binds to a target molecule of interest.[149] Some RNA aptamers evolved naturally as riboswitches that bind and respond structurally to intracellular metabolites, thereby regulating gene expression.[150] However, most interest in aptamers focuses on the ability to rapidly evolve and select for aptamers that bind novel targets through the process of systematic evolution of ligands by exponential enrichment (SELEX).[151] By incubating a starting pool of 10^{12}-10^{15} diverse oligonucleotides with magnetic nanoparticle-bound or otherwise immobilized target molecules, then performing successive rounds of washing, PCR amplification, target binding, and separation from the pool, researchers have been able to select for aptamers that bind to a vast range of molecular targets, from proteins[152] to small molecules[153] to monoatomic ions.[154] These aptamers have subsequently been used to build biosensors for the electrochemical,[155] fluorescent,[156] and colorimetric[157] detection of their target molecules. Sometimes called 'molecular antibodies,' aptamers have several theoretical advantages over their immunoglobulin counterparts: they are molecularly defined and can be chemically synthesized, potentially increasing reproducibility; they are much more stable and have longer shelf lives than antibodies, potentially increasing their utility for 'field' diagnostics;[158] and finally, with the right set of negative selection steps during SELEX, aptamers can be rapidly evolved to discriminate between structurally similar target molecules with low cross-reactivity.[159] In one relevant example, researchers reported evolving a library of aptamers that bind specifically to one steroid hormone (including DHEA-S and cortisol) with nanomolar affinities and without binding to other steroid hormones.[160]

Over the past two decades, Mirkin lab has developed numerous nanoparticle-based assays of disease biomarkers, including nucleic acids,[71] proteins,[74] and small molecules.[72] One of the most well-developed is the NanoFlare,[77] a gold nanoparticle (AuNP) SNA functionalized with thiolated DNA oligonucleotides that are hybridized to fluorophore-modified oligonucleotide 'flare' strands. Proximity to the AuNP quenches

the fluorophores on the flare strand; however, in the presence of a target polynucleotide sequence that is complementary to the thiolated oligonucleotide, the target hybridizes to the SNA and displaces the flare strand. The displaced flare diffuses away from the AuNP and is no longer quenched, generating a fluorescent signal. NanoFlares represent a tailorable platform for the simple and rapid detection of numerous biomarkers. Because SNAs readily enter cells, NanoFlares can detect nucleic acid targets both extracellularly and intracellularly.[79] Multiplex NanoFlares which can simultaneously detect two different target sequences using two different thiolated oligonucleotides and two flare strands with different fluorophores conjugated to them have already been demonstrated, opening the possibility of one-pot analysis of multiple biomarkers using NanoFlares.[78]

Aptamer NanoFlares are an intriguing and underexplored class of NanoFlares (**Figure 16**).[81] Aptamer NanoFlares are AuNPs functionalized with a thiolated DNA aptamer that binds to a target molecule of interest an contains a hairpin stem. The flare strand is hybridized onto the hairpin stem, displacing the

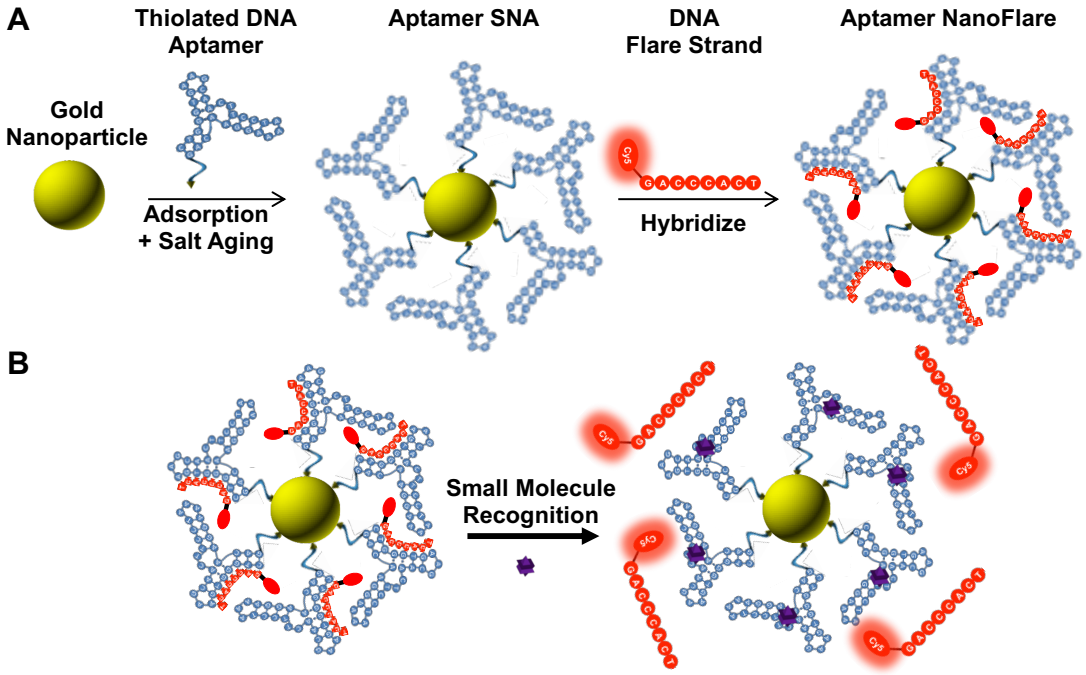


Figure 16. Aptamer NanoFlares for small molecule detection. (A) Synthesis of Aptamer NanoFlares from citrate-capped gold nanoparticles, thiolated DNA Aptamers, and complementary fluorophore modified Flare strands. (**B**) Small molecule recognition with aptamer NanoFlares by stabilizing the folded aptamer and/or destabilizing the aptamer-flare duplex.

aptamer's self-complementary duplex and partially disrupting the aptamer's structure. Then, in the presence of the target molecule, the folded conformation of the aptamer is stabilized, re-forming the self-complementary duplex and displacing the flare strand, which diffuses away from the AuNP and generates a fluorescent signal. The first demonstrated aptamer NanoFlare bound to and detected ATP *in vitro*, could distinguish ATP from other nucleoside triphosphates in a one-pot reaction, and could enter cells and detect when ATP production was halted.[81] However, few if any subsequent aptamer NanoFlare designs have been reported, meaning that the generalizability of this biosensor architecture is unknown. Moreover, the first aptamer NanoFlare paper demonstrated biosensing function without delving deeply into the mechanisms and design considerations undergirding that function. Whether and how the behavior and the design rules for minimizing the detection limit of aptamer Nanoflares differ from either nucleic acid-sensing NanoFlares or nanoparticle-free aptamer biosensors remains unknown.

To address these questions, this chapter presents our efforts to model, measure, and screen for aptamer NanoFlares to detect DHEA-S and cortisol biomarkers of stress.

3.2 Initial Aptamer NanoFlare design

To begin building and testing aptamer NanoFlares for detection of biomarkers of stress, we selected as a first model the aptamer DIS11th_3, which had been designed by running SELEX with 7 rounds of positive selection for DHEA-S binding, then 3 rounds of counterselection against several non-DHEA-S steroids, and then one final round of positive DHEA-S binding selection.[160] This aptamer has a predicted secondary structure with three hairpins, and the DHEA-S is predicted to bind at the juncture of the three hairpins. To adapt the aptamer for functionalization on an SNA, a 7-adenosine (7A) spacer and then a thiol modification were added to the 3' end of the aptamer. The initial flare strand was designed

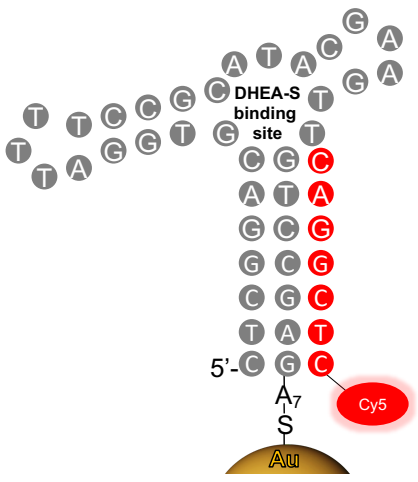


Figure 17. Architecture of initial DHEA-S aptamer NanoFlare design.

to hybridize to the 8 base pairs on the 3' end of the aptamer, theoretically bringing the flare's 5' Cy5 modification into close proximity to the AuNP core of the SNA and quenching its fluorescence (**Figure 17**).

Using this design, aptamer NanoFlares were synthesized, and their ability to detect DHEA-S was tested. The initial aptamer NanoFlare design did not display an increase in fluorescence in the presence of 1 mM DHEA-S (**Figure 18**).

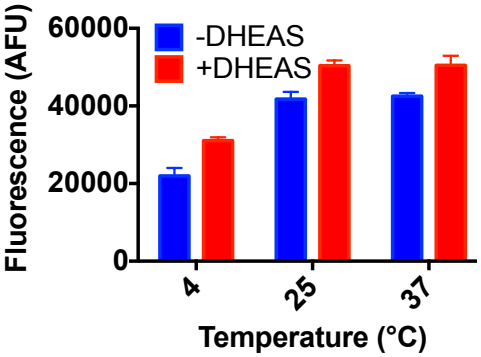


Figure 18. Initial DHEA-S aptamer NanoFlare design does not detect DHEA-S. Fluorescent response of aptamer NanoFlare is shown in the presence (red) and absence (blue) of 1 mM DHEA-S.

3.3 DIS11th_3T truncated aptamer NanoFlare design.

The increasing background signal with increasing temperature in this aptamer NanoFlare design suggests that the flare strand does not hybridize stably, possibly due to the competing internal duplex that is formed when the aptamer is folded. We hypothesized that a shorter, less stable aptamer hairpin might function better, because it would be more accessible to hybridization with the flare strand. We therefore designed a truncated version of the DIS11th_3, named DIS11th_3T, which shortened the hairpin stem by 4 base pairs (**Figure 19**). In this design, the flare strand now is longer than the aptamer's self-complementary duplex, and extends into the DHEA-S binding site. After synthesizing aptamer NanoFlares with this design, a preliminary DHEA-S detection experiment indicated that DIS11th_3T could detect high (1 mM) concentrations of DHEA-S (**Figure 20**).

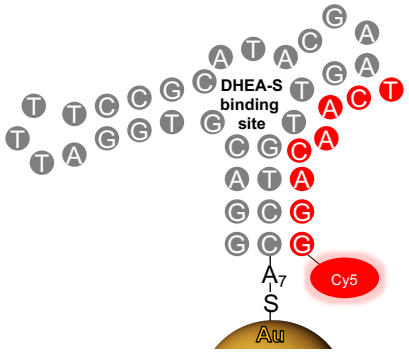


Figure 19. Truncated aptamer NanoFlare design.

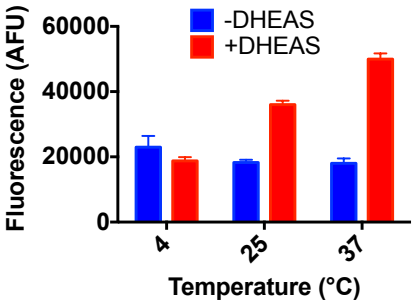


Figure 20. Truncated aptamer NanoFlare design may detect DHEA-S. Fluorescent response of aptamer NanoFlare is shown in the presence (red) and absence (blue) of 1 mM DHEA-S.

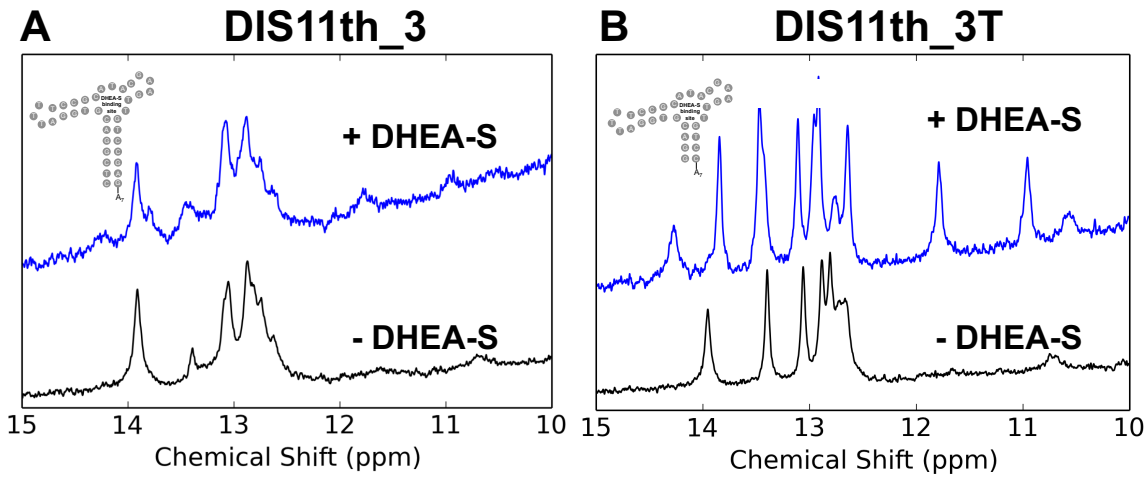


Figure 21. **NMR of aptamer structural response to DHEA-S.** (**A**) Imino proton NMR spectra for the original DIS11th_3 aptamer in the presence and absence of a 1.25-fold excess of DHEA-S. (**B**) NMR spectra of DIS11th_3T in the presence and absence of 1.25X DHEA-S.

3.4 Structural response of DIS11th_3 and DIS11th_3T to DHEA-S.

We sought to test the hypothesis that the truncated aptamer's structure was destabilized relative to the original DIS11th_3. In collaboration with Peter Mirau's lab, we generated NMR spectra for the two aptamers in the presence and absence of DHEA-S (**Figure 21**). The imino protons in unpaired guanine and thymine nucleobases rapidly exchange with solvent, such that they do not generate measurable NMR peaks. However, in G-C and A-T base pairs, these imino protons are protected from solvent, and can appear as peaks on the NMR spectra. The original DIS11th_3 aptamer has largely the same spectrum in the presence and absence DHEA-S, indicating that the presence of target molecule does not change the structure of the aptamer. However, for the truncated DIS11th_3T aptamer, the presence of DHEA-S causes at least 3 new peaks, at 11 ppm, 12 ppm and 14.5 ppm, to appear on the NMR spectrum. This indicates that DHEA-S induces a conformational change in DIS11th_3T, which is promising from the standpoint of designing a biosensor that responds structurally to the presence of target molecule.

3.5 Backfilling aptamer NanoFlares to reduce nonspecific quenching

In order to efficiently probe parameters governing aptamer Nanoflare performance, it was important to establish the minimum background and maximum fluorescence signal the particles can produce. In

particular, we hypothesized that inefficient binding/hybridization of the flare strand to the aptamer SNA might increase background fluorescence and reduce the dynamic range and signal to noise ratio of the biosensor. To determine how efficiently the particles quench flare fluorescence, and whether that quenching is due to hybridization with the aptamer stem, we measured the fluorescence from DHEA-s flare strands alone, flares in the presence of gold nanoparticles, and flares in the presence of either the DIS11th_3T aptamer SNA or MN19,[161] a non-complementary cocaine aptamer SNA (**Figure 22**; sequences in **Table A1**).

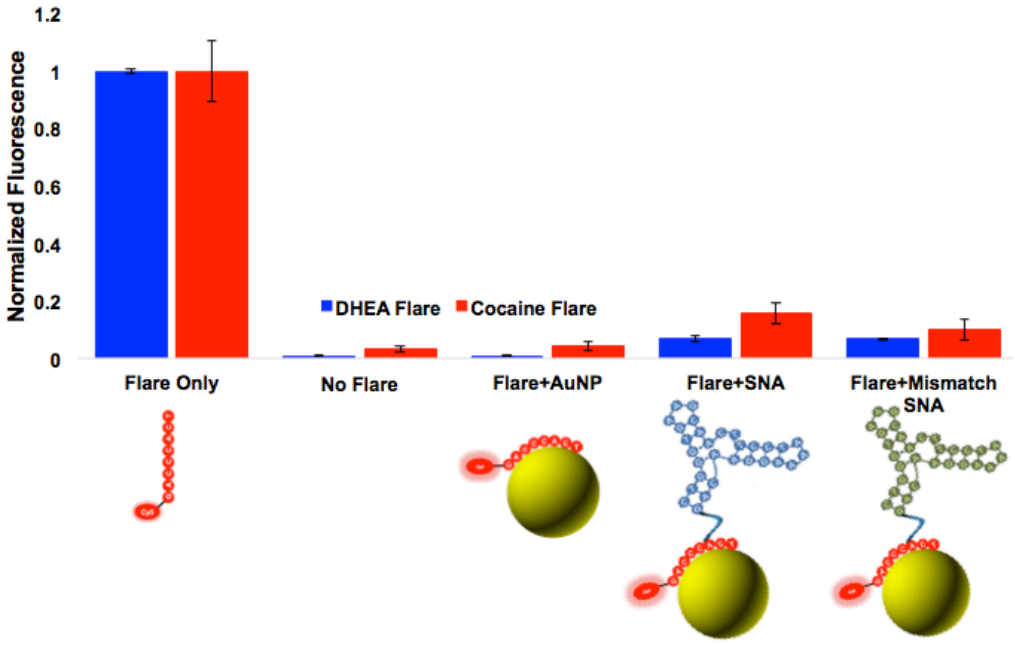


Figure 22. Fluorescence and quenching mechanism for flare strands. Both DHEA-S and cocaine flares were incubated in PBS at 37ºC in the presence of bare gold nanoparticle, matching aptamer SNA (i.e. DHEA flare with DHEA aptamer), or with a mismatched aptamer SNA (i.e. cocaine flare with DHEA aptamer).

The nanoparticles efficiently quenched the flare strand, generating a >10-fold decrease in fluorescence relative to free flare. However, the quenching appeared to be nonspecific: free AuNPs with no aptamer strands attached and non-complementary aptamer SNAs both efficiently quenched the flare strand, indicating that hybridization with the aptamer stem is not required for the flare strand to be quenched. Free DNA strands are known to interact nonspecifically with AuNPs,[162] so we suspected that most of the flare strands were adsorbing onto the surface of the gold nanoparticles in the spaces between the aptamer strands.

We hypothesized that disrupting the flare strand's ability to access the gold nanoparticle's surface would reduce non-specific quenching of the flare. We tested the effect of backfilling the aptamer-SNA particles with thiolated PEG oligomers (average MW = 800 Da), and measured the change in fluorescence (**Figure 23**).

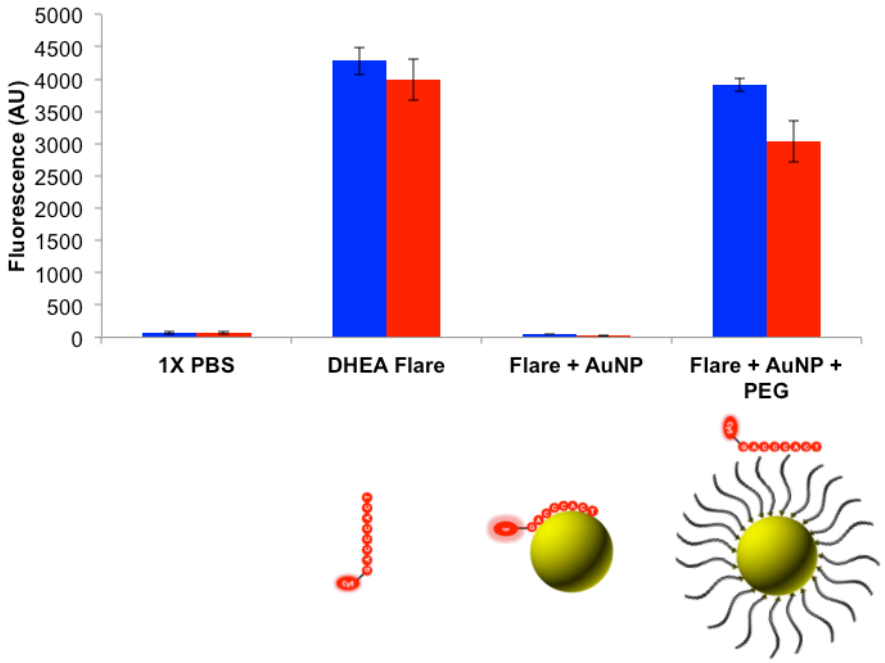


Figure 23. Backfilling Aptamer SNAs with PEG greatly reduces non-specific quenching. Both DIS11th_3T and MN19 flare strands were incubated by themselves, with bare AuNPs, or with gold nanoparticles incubated with an excess of thiolated PEG oligomers (1,000:1 PEG:AuNP).

Functionalizing the surface of the bare gold nanoparticles with PEG almost completely eliminated quenching of either flare strand. This suggests that the nonspecific quenching was due to adsorption of the flare strand with the gold nanoparticle's surface.

We next sought to determine whether the backfilled aptamer-SNAs would still hybridize and quench their flare strands in a sequence-dependent manner (**Figure 24**). We also sought to measure whether how much of a change in fluorescence could be induced by de-hybridizing the flare strands off the particle. We therefore incubated DHEA aptamer SNAs with either DHEA flare strands or cocaine flare strands, and measured fluorescence in the presence and absence of urea.

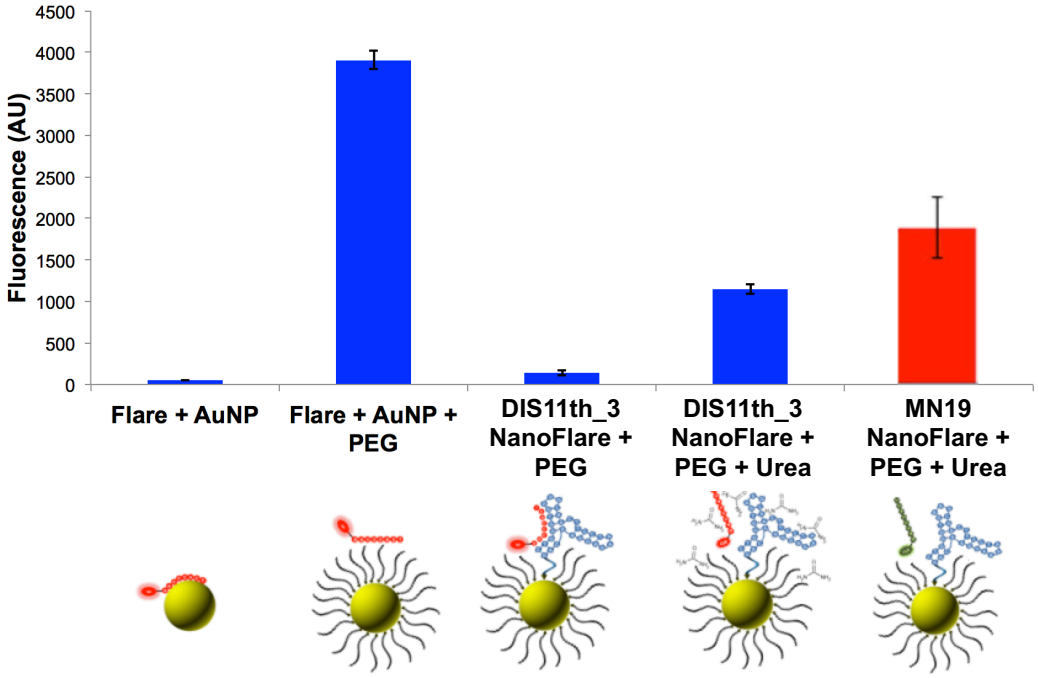


Figure 24. Backfilled Aptamer Nanoflares demonstrate reversible, sequence-dependent quenching. DIS11th_3 flare strand (blue bar) was incubated with bare gold nanoparticle, backfilled gold nanoparticle, backfilled DIS11th_3 aptamer SNA, and backfilled DIS11th_3 aptamer SNA in the presence of urea. MN19 cocaine flare (red bar) was also incubated with backfilled DIS11th_3 aptamer SNA.

While PEGylated AuNPs did not efficiently quench the DHEA flare strand, the PEGylated DIS11th_3 aptamer SNA did. This indicated that the flare strand interacts with the DIS11th_3 aptamer SNA via a distinct mechanism from the bare gold nanoparticle interaction. Further, the fluorescence of the backfilled DIS11th_3 NanoFlare increased more than 8-fold upon addition of urea to the solution, suggesting that denaturation and de-hybridization destroyed the flare strand-aptamer SNA interaction. Finally, the cocaine flare strand was not efficiently quenched by the backfilled DIS1th_3 aptamer SNA, suggesting that the interaction of the flare strand with backfilled nanoparticles was sequence-dependent.

3.6 Measuring flare hybridization efficiency

We wanted to determine how well backfilled aptamer Nanoflares could detect small molecules of interest. Surprisingly, we observed no increase in fluorescence when DIS11th_3 aptamer NanoFlares were added to solutions with 1 mM DHEA-S. We suspected that non-hybridized flare strands in solution may be

causing high background fluorescence levels, thus impairing detection. We therefore synthesized aptamer Nanoflares with different numbers of flare strands per particle, hypothesizing that some optimal number of flare strands would provide the lowest background fluorescence levels. However, none of the particles we synthesized detected DHEA-S (**Figure 25**).

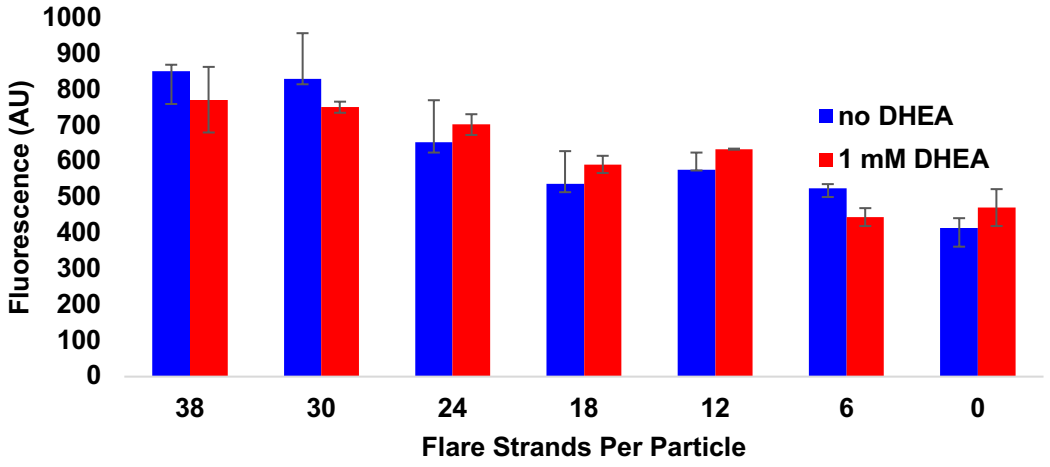


Figure 25. Flare Loading versus DHEA-S detection. DIS11th_3 aptamer NanoFlares are synthesized with different numbers of flare strands per particle. Then fluorescence is measured with (red) and without (blue) 1 mM DHEA-S.

Hypothesizing that inefficient flare quenching may reduce aptamer Nanoflare performance, we sought to measure how efficiently the DIS11th_3 SNAs quench increasing numbers of DHEA-S flare strands. We measured the fluorescence of Aptamer Nanoflares with different ratio of aptamer strands to flare strands,

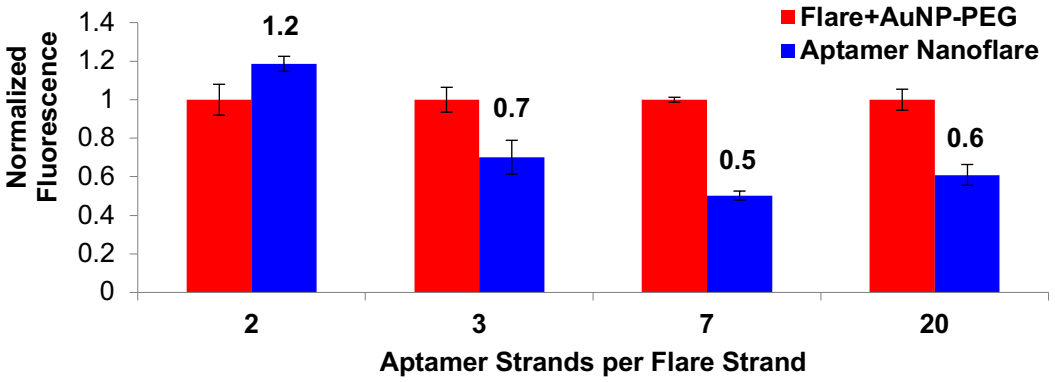


Figure 26. Flare loading versus flare quenching. Aptamer Nanoflares were synthesized with varying loadings of flare strand, expressed as a ratio of aptamer strands to flare strand. The aptamer Nanoflare fluorescence (blue) is compared to the fluorescence of PEGylated gold nanoparticles mixed with the same number of flare strands, and the ratio of the two is labeled.

compared to PEGylated AuNPs with the same number of flare strands (**Figure 26**). We found that NanoFlares with 7 aptamer strands per flare strand showed the most efficient quenching, causing a 2-fold decrease in fluorescence. As 2-fold quenching is much less efficient than has been observed for other NanoFlare systems, we re-synthesized the aptamer, flare, and scrambled flare sequences and repeated the experiment with fresh reagents. We again observed at most a 2-fold quenching effect, with most efficient quenching observed at a ratio of 10 aptamer strands per flare strand.

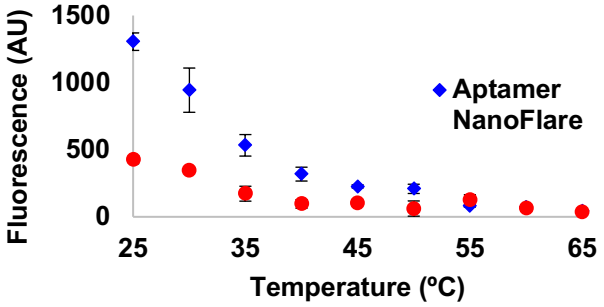


Figure 27. DIS11th_3 aptamer NanoFlare Melt (Plate Reader). The fluorescence of Aptamer NanoFlares (blue) and flare strands by themselves (red) is measured as temperature rises from 25ºC to 65ºC.

We hypothesized that the flare strands may be quenched inefficiently because they melt off the SNA at low temperatures. We therefore ran a melting experiment on the aptamer Nanoflares in the plate reader, measuring the fluorescence of the constructs as we raised the temperature from 25ºC to 65ºC (**Figure 27**). Surprisingly, we observed that aptamer Nanoflare fluorescence decreased with increasing temperature, in contrast to what would be expected for a fluorophore-labeled oligonucleotide hybridized onto a gold nanoparticle. The fluorescence of free flare strands alone in solution also decreased in fluorescence with increasing temperature.

This led us to hypothesize that the DIS11th_3 flare strand-aptamer duplex had an even lower melting temperature

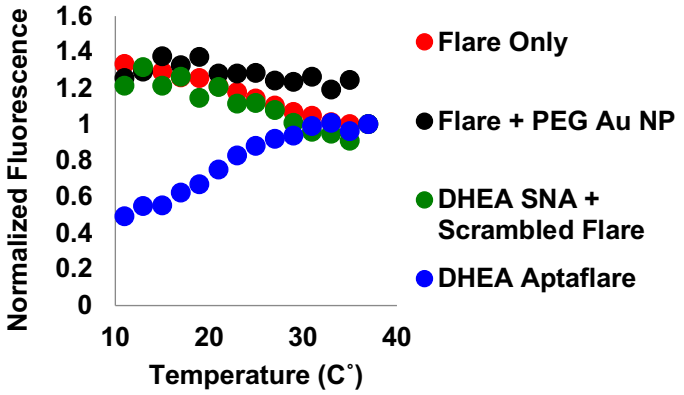


Figure 28. Flare strand annealing. 1 nM DIS11th_3 aptamer NanoFlares ('DHEA Aptaflare', blue) are slowly cooled in a fluorimeter from 37ºC to 9ºC. Control samples including flare only (red), flare + PEGylated gold nanoparticle (black), and DIS11th_3 SNA (DHEA SNA) + scrambled flare (green) are shown.

than 25ºC. We therefore used a fluorimeter to measure annealing of the flare strand to the particle as the temperature was slowly (1ºC/minute) lowered from 37ºC to 9ºC (**Figure 28**). Under these conditions, fluorescence of the aptamer Nanoflare constructs decreased 2-fold between 30ºC and 15ºC. We suspected that the flare strands had a relatively low melting temperature because they were only 8 base pairs long. We concluded that the stability of the flare-aptamer duplex needed to be increased in order to increase the potential signal the aptamer NanoFlares were capable of generating.

3.7 Increasing flare strand length

We hypothesized that a longer flare that more strongly hybridized to the aptamer stem would be quenched more efficiently, leading to lower baseline fluorescence in the aptamer NanoFlare biosensor. We therefore designed and synthesized a 12 base flare strand (**Figure 29**), which is 4 bases longer and has a predicted melting temperature 20ºC higher than the original 8 bp DIS11th_3T flare (40ºC versus 20ºC). We also synthesized fresh aptamer SNAs, achieving a loading density of 40 aptamers/AuNP as measured by Oligreen.

Short Flare: 5'-Cy5-GGACAACT-3' **Predicted T_M = 20ºC**
Long Flare: 5'-Cy5-GGACAACTTCGT-3' **Predicted T_M = 40ºC**

Figure 29. Original, shorter 8 bp DIS11th_3 flare sequence (red), and longer 12 bp DIS11th_3 flare sequence (blue). Melting temperatures predicted the IDT Oligo Analyzer tool are shown.

We then compared the quenching efficiency of the 12 bp and 8 bp flare strands in a fluorimeter melt experiment (**Figure 30**). While the 12 bp flares achieved 3-fold quenching of flare strand below 30ºC, the 8 bp flare strands displayed little quenching and a higher background even at 10ºC. This experiment demonstrated that NanoFlares made with the 12 bp flare strand quenched baseline fluorescence more efficiently and at higher temperatures than previous constructs made with the 8 bp flares. The aptamer NanoFlares appeared to detect DHEA-S.

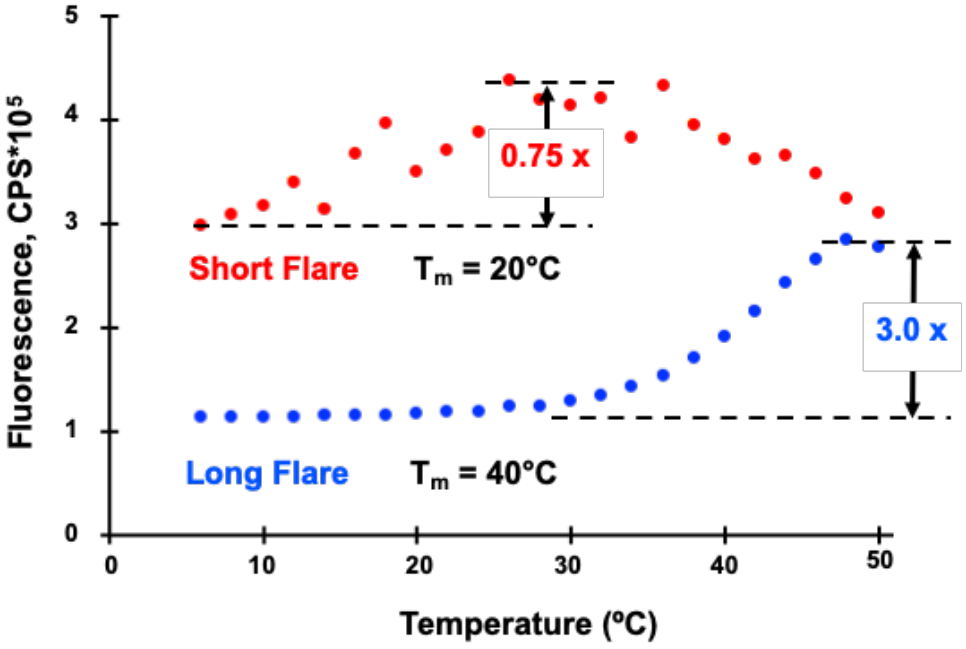


Figure 30. Fluorimeter melt of 8 bp flare strand (red) versus 12 bp flare strand (blue) on backfilled aptamer SNAs.

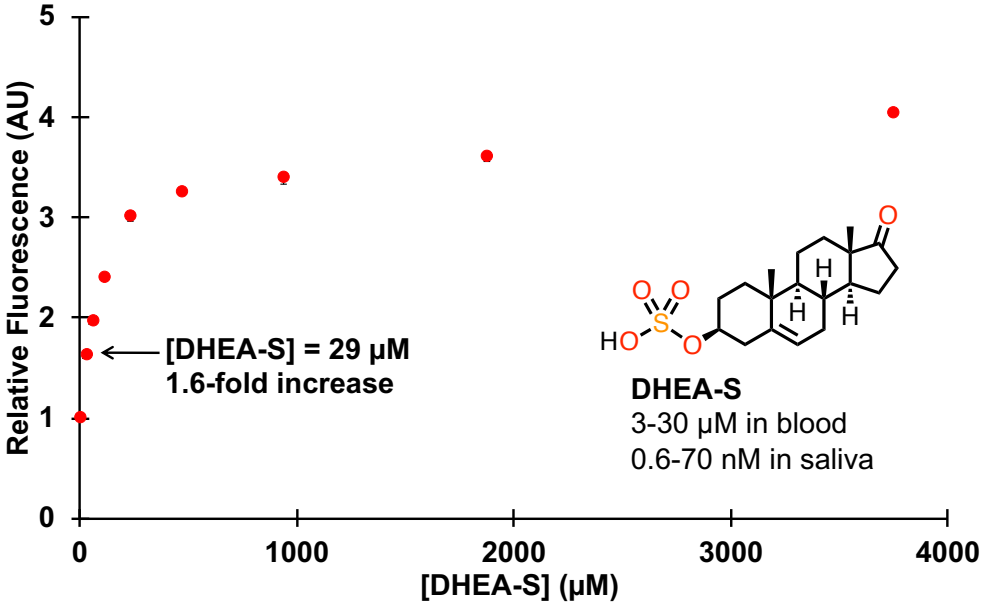


Figure 31. DHEA-S detection over a range of concentrations. Aptamer Nanoflare fluorescence was measured in a fluorimeter in the presence of a range of DHEA-S concentrations. Physiological concentrations of DHEA-S in blood and saliva are also shown.

3.8 DHEA-S detection and batch-to-batch variability

We next tested whether aptamer NanoFlares with the 12bp flare strands could detect DHEA-S. The fluorescence of aptamer NanoFlares was measured in the presence of a dilution series of DHEA-S concentrations, ranging from 3.75 mM to 30 μM DHEA-S (**Figure 31**). Fluorescence increased for all concentrations of DHEA-S, increasing roughly linearly in proportion to the logarithm of DHEA-S concentration. In this experiment the aptamer NanoFlares appeared to detect DHEA-S.

However, subsequent experiments have not consistently replicated this DHEA-S detection curve. For example, we synthesized several batches of aptamer NanoFlares with a variety of loading densities, ranging from 88 aptamers/particle to 56 aptamers/particle, in order to test the effect of aptamer loading density on DHEA-S detection **(Figure 32)**. The resulting detection curves showed a wide variation in fluorescence response between particle batches that did not correlate with loading density: while particles with the highest loading density (88 aptamers/particle) showed the greatest relative increase in fluorescence, the second greatest fluorescent response came from particles with the lowest loading density (56 aptamers/particle). The two particles with intermediate aptamer loading densities showed the lowest fluorescent response. The source of this batch-to-batch variability is still unclear.

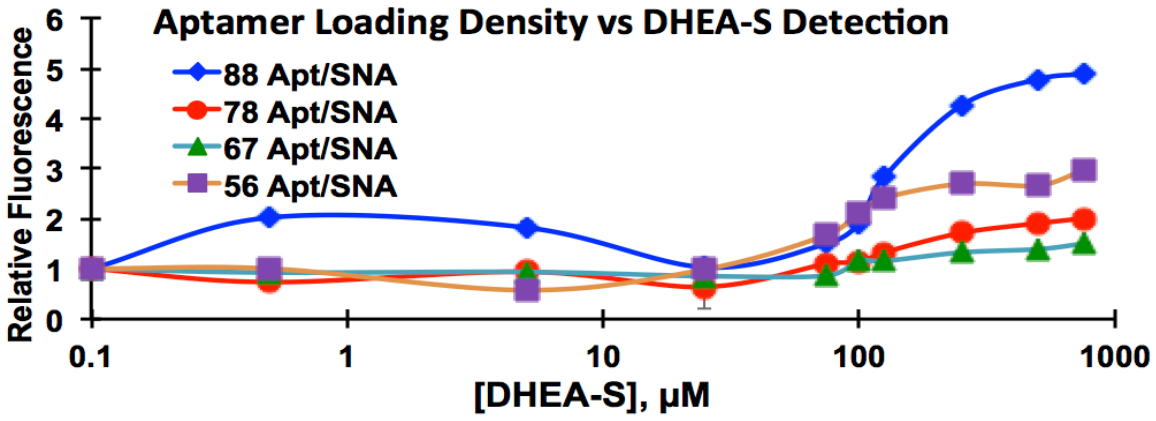


Figure 32. Detection of DHEA-S by multiple batches of aptamer Nanoflares. Plate reader data of the fluorescence response to a range of DHEA-S concentrations by aptamer NanoFlare batches with different loading densities. Experimental data is in triplicate.

3.9 Conformational selection model of aptamer NanoFlares

In order to move beyond a specific aptamer NanoFlare system and seek to improve the design and performance of aptamer NanoFlares in a generalizable way, we sought to build, validate and refine mathematical models of aptamer NanoFlares. In collaboration with Peter Mirau, we first designed a 3-state conformational model of the aptamer NanoFlare system (**Figure 33**).

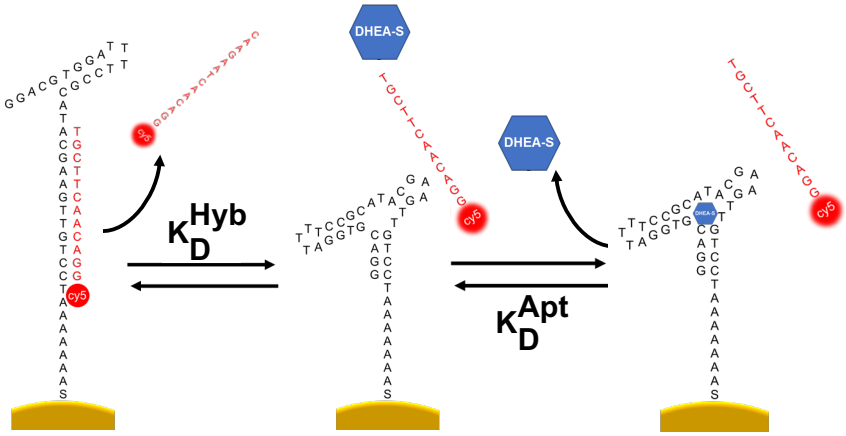


Figure 33. 3-state conformational selection model of the aptamer Nanoflare system. The unfolded aptamer and hybridized flare strand (on the left) exist in equilibrium with the folded aptamer and dissociated flare (in the center), and in which the addition of DHEA-S stabilizes the folded state of the aptamer (on the right).

Five equations define the aptamer Nanoflare 3-state model (**Figure 34**). Three equations state that the overall concentration of aptamer strand, DHEA-S small molecule, and flare strand, are constant during each experiment. These overall concentrations are known and set by the initial conditions of each experiment, while the concentrations of free flare strand, aptamer, DHEA-S, flare-aptamer duplex and DHEA-S-aptamer complex are unknown. The most important parameters in the model are the two dissociation constants (K_D's). The dissociation constant of the aptamer-DHEA-S interaction (K_D^{Apt}) influences how much the presence of small molecule stabilizes the aptamer. The dissociation constant of the aptamer-flare strand hybridization

$$(1)\ K_D^{Hyb} = \frac{[F][A]}{[AF]} \qquad (3)\ [A] + [AD] + [AF] = A_{Tot}$$

$$(4)\ [F] + [AF] = F_{tot}$$

$$(2)\ K_D^{Apt} = \frac{[A][D]}{[AD]} \qquad (5)\ [D] + [AD] = D_{Tot}$$

Figure 34. Equations of the conformational selection aptamer NanoFlare model. Equations 1 and 2 define the dissociation constants of the aptamer-flare and aptamer-DHEA-S interaction. Equations 3, 4 and 5 define the fixed concentration of aptamer (A), flare strand (F), and DHEA-S (D) in each experiment.

interaction (K_D^{Hyb}) influences how efficiently the flare strand hybridizes, and therefore the background fluorescence of aptamer NanoFlares. Determining these dissociation constants will result in a set of 5 equations with 5 unknowns, which can be solved to generate predictions about aptamer NanoFlare behavior. Using Jupyter Notebooks, we built a Python-based version of this equilibrium, conformational selection model (**Code C1**).

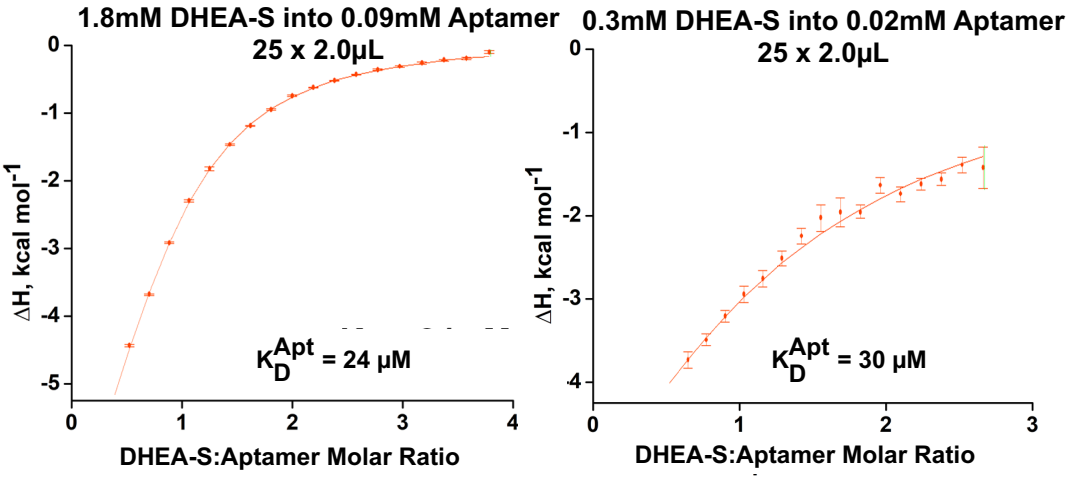


Figure 35. Measuring K_D^{Apt} with isothermal titration calorimetry. The ΔH released from the titration of DHEA-S is plotted as a function of the molar ratio of DHEA-S to Aptamer. Two different titration curves, plotting either titration of 1.8 mM DHEA-S into 0.09 mM aptamer (**A**) or titration of .3 mM DHEA-S into .02 mM aptamer (**B**). The calculated K_D^{Apt} from each titration curve is shown and the average K_D^{Apt} of the two measurements (bottom) is shown.

3.10 Determining aptamer-flare and aptamer-DHEA-S dissociation constants.

We performed isothermal titration calorimetry to measure K_D^{Apt} (**Figure 35**), measuring the heat released as DHEA-S was titrated into a solution of free aptamer in 1X PBS. A curve was then fitted to the ΔH vs DHEA-S concentration. We ran this experiment with two different concentrations of DHEA-S and aptamer to make sure the measurement was consistent, and obtained an average K_D^{Apt} of 27 μM ± 2 μM.

We hypothesized the value of K_D^{Hyb} could be inferred by comparing the predicted behavior of the model for a range of K_D^{Hyb} values to the experimentally observed behavior of the particles. Toward this end, we compared the experimentally observed flare quenching efficiency from the fluorescence melt in **Figure 30** to the model's predicted quenching efficiency for a range of K_D^{Hyb} values **(Figure 36)**. For 1 nM Aptamer SNA (40 nM aptamer) and 4 nM flare strand in 1X PBS, fluorescence at 30ºC was 66% lower than

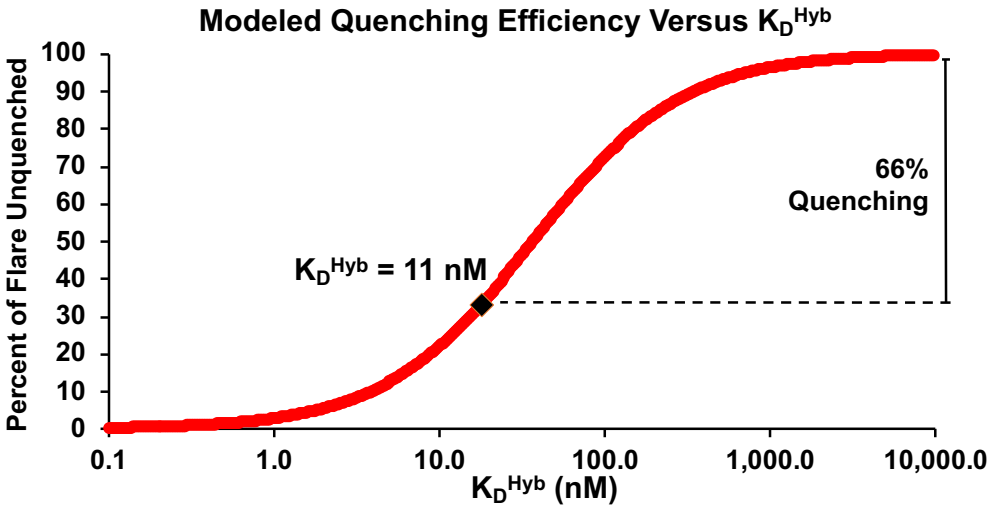


Figure 36. Model-based inference of K_D^{Hyb}. Model prediction of fraction of free flare strand (equal to 1 minus the quenching efficiency) versus different values of K_D^{Hyb}.

fluorescence at 50ºC. The model requires a K_D^{Hyb} of 11 nM in order to reproduce 66% quenching of flare strands at these concentrations of aptamer and flare strand.

3.11 Predictions of equilibrium conformational selection model

Having determined the dissociation constants required to solve the conformational selection mathematical model, we sought to use it as a guide for future optimizations of aptamer NanoFlare designs and assays. Three main model parameters could affect assay performance: the concentration of aptamer Nanoflare, the strength of flare-aptamer binding (K_D^{Hyb}), and the strength of the aptamer-DHEA-S interaction (K_D^{Apt}). We plotted model predictions about the relative fluorescence response to DHEA-S for a range of aptamer NanoFlare concentrations (**Figure 37**). The model predicted that higher concentrations of aptamer Nanoflare would lead to larger relative fluorescence increases in the presence of similar concentrations of DHEA-S. For instance, 1 nM particle was predicted to only generate a maximum 3.5-fold increase, 10 nM particle was predicted to generate up to a 20-fold fluorescence increase in the presence of 10 mM DHEA-S. This made sense, because mass action meant that higher concentrations of aptamer and flare would translate to a higher proportion of flare hybridized onto the aptamer SNA and quenched. Therefore, a larger proportion of flare strands could be displaced by high concentrations of DHEA-S.

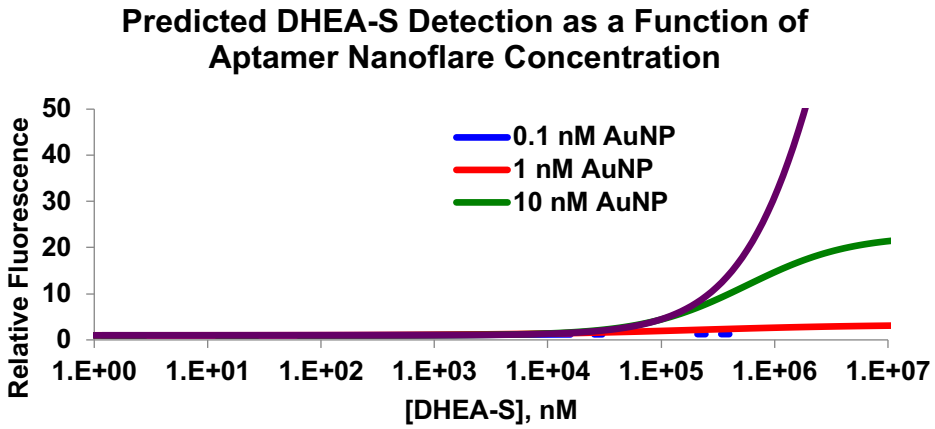


Figure 37. Predicted DHEA-S detection as a function of aptamer NanoFlare concentration. Aptamer Nanoflare concentrations are expressed as gold nanoparticle concentrations with 30 aptamers per particle, and 5 flare strands per particle.

However, higher concentrations of DHEA-S did not translate to a lower predicted detection limit; approximately the same concentration of DHEA-S was required to generate a predicted 1.5-fold fluorescence increase in 1 nM aptamer Nanoflare as was needed at 10 nM or 100 nM aptamer NanoFlare.

Next, we examined how varying the value of K_D^{Hyb} affected the model's behavior (**Figure 38**). Perhaps unsurprisingly, lower K_D^{Hyb} values were predicted to produce larger relative fluorescence changes in the presence of DHEA-S. For instance, while an 11 nM K_D^{Hyb} was predicted to generate at most a 3.5-fold fluorescence increase, a 1 nM K_D^{Hyb} was predicted to generate up to a 25-fold fluorescence increase. This

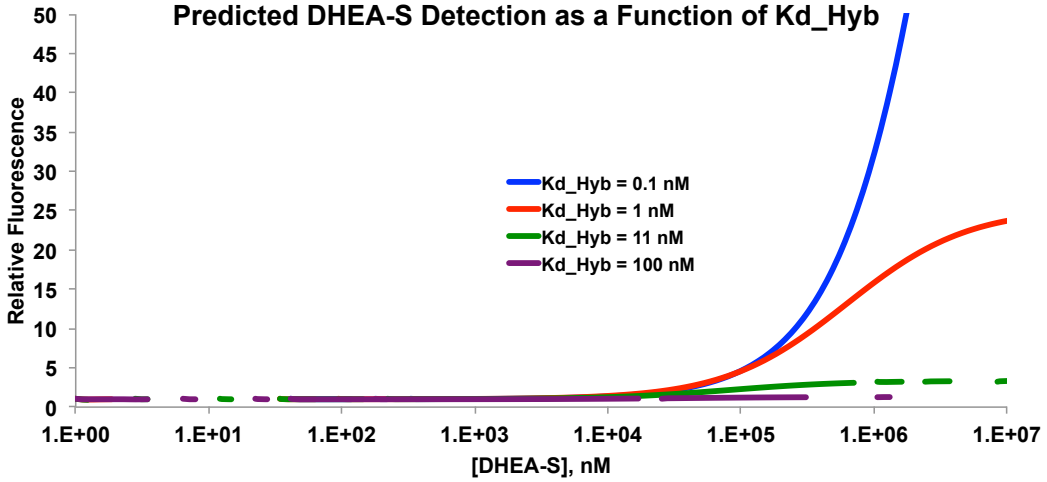


Figure 38: **Predicted DHEA-S detection as a function of K_D^{Hyb}.**

was because tighter binding flares more efficiently hybridize and quench aptamer SNAs. Importantly, however , the model assumed no noise in the fluorescence measurement, so the largest predicted fold changes in fluorescence may come from unrealistically low predictions about background noise and baseline fluorescence. As with varying the construct's concentration, the higher relative fluorescence with lower K_D^{Hyb} did not translate to a decrease in the predicted detection limit of the assay—the amplitude of the sigmoid fluorescent response to DHEA-S increased, but the sigmoid did not shift to lower concentrations of DHEA-S.

We then plotted predicted DHEA-S detection performance as a function of the strength of aptamer-DHEA-S binding, K_D^{Apt} (**Figure 39**). This was the only parameter that was clearly predicted to affect the detection limit of the aptamer NanoFlares: the stronger the aptamer-small molecule interaction (and therefore the lower the K_D^{Apt}), the more the detection response curve shifted to lower concentrations of DHEA-S. The sigmoid fluorescent response also became sharper at lower K_D^{Apt} values, because most of the DHEA-S in solution would bind to the aptamer and displace flare strands. These results suggested that significantly reducing the detection limit of aptamer Nanoflares would requires optimization of the aptamer structure to bind more tightly to the target molecule. However, K_D^{Apt} has no effect on the amplitude of the fluorescent response to large concentrations of DHEA-S.

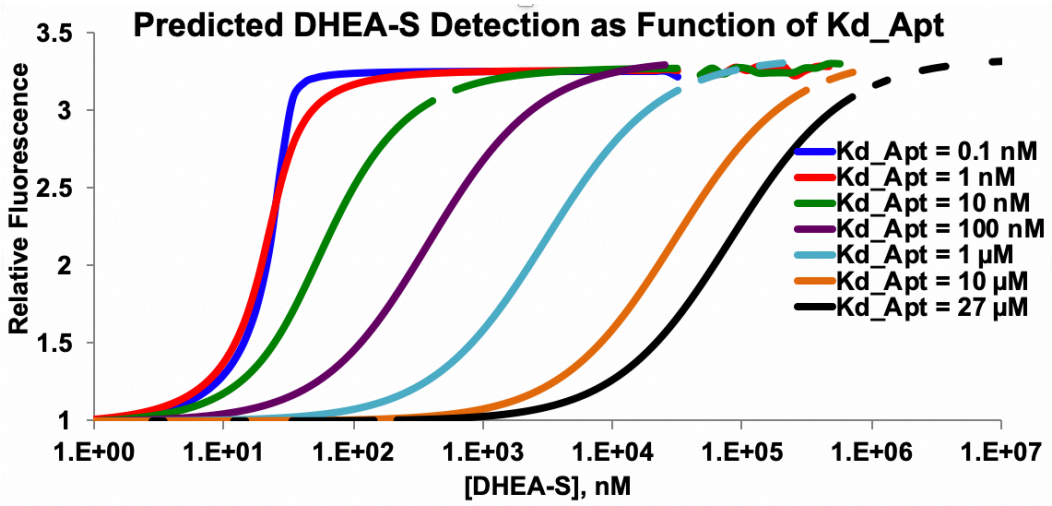


Figure 39: Predicted DHEA-S detection as a function of K_D^{Apt}. Predicted detection curve with experimentally determined K_D^{Apt} (27 µM) in black.

3.12 Induced fit kinetic model of aptamer NanoFlares

The conformational selection model assumed that the flare and aptamer dynamically associate and dissociate, that DHEA-S stabilizes the folded, flare-inaccessible conformation of the aptamer, and that the equilibrium between flare-aptamer duplex and aptamer-DHEA-S complex is governed by the dissociation constants of the two interactions (K_D^{Hyb} and K_D^{Apt}) and the concentrations of the three molecules. However, we hypothesized that the assumption that flare strands rapidly and dynamically hybridize and dissociate from the SNA was not accurate, particularly given that SNAs are known to hybridize more stably to complementary DNA than equivalent linear oligonucleotides.[61] We investigated an alternative, induced fit model of aptamer Nanoflare dynamics (**Figure 40**). In an induced fit regime, the flare strand hybridizes stably with the aptamer and its dissociation rate (k_R^{Hyb}) is low. To displace the flare strand, the target molecule binds to the flare-aptamer duplex and induces it to change shape, into an intermediate structure in which the flare dissociation rate (k_R^{Hyb*}) is higher, and de-hybridization becomes more favorable.

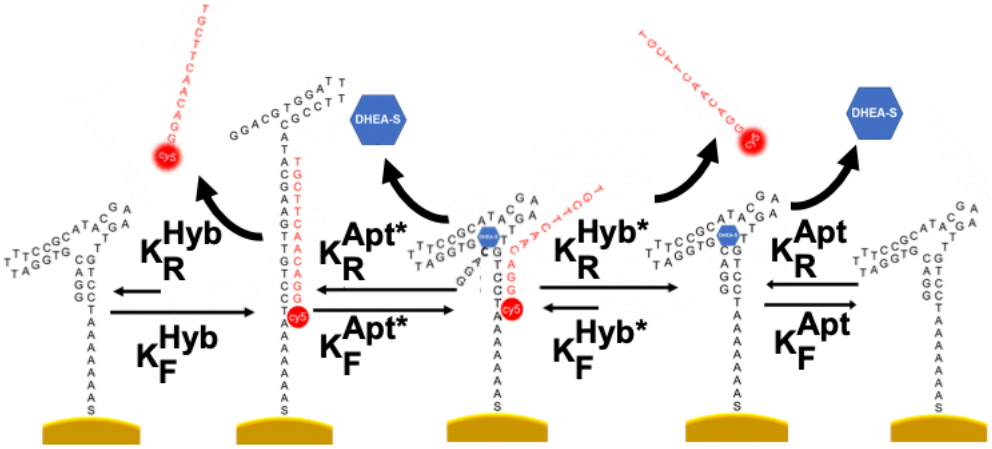


Figure 40. Induced fit model of aptamer Nanoflares.

One observable difference between the conformational selection and induced fit models is that in an induced fit system, the flare strand rarely dissociates in the absence of the aptamer's target molecule. To test hybridized flare strands' tendency to dynamically dissociate from the aptamer, DIS11th_3T aptamer NanoFlares with 12 bp flare strands were incubated in PBS at room temperature for 15 minutes and pelleted via centrifugation, and then the supernatant was pipetted off and measured for fluorescence. This was

repeated five times, and then the pellet was resuspended in 2 M urea, incubated and pelleted once more (**Figure 41**). If the DIS11th_3T aptamer NanoFlare system were governed by conformational selection dynamics, then some fraction of the flare strands would be expected to dissociate with each cycle of pelleting and resuspending in fresh buffer, leading to fluorescence in the supernatant. However, if an induced fit model better described aptamer NanoFlare behavior, little to no flare would be measured in the supernatant, even after repeated rounds of pelleting, removing the supernatant, and washing with fresh buffer. Fluorescence would appear in the supernatant only after, for instance, denaturing the aptamer-flare duplex with concentrated urea. This is precisely what is observed in the pelleting experiment.

The pelleting results suggested that the aptamer NanoFlares had a very low aptamer-flare dissociation rate: the amount of flare dissociating in each round of washing and 30 minutes of incubation was at most 1/120th of the total flare hybridized on the particle. This result was interesting for three primary reasons: first, pelleting could be a useful strategy when optimizing aptamer nanoflare response to low concentrations of target molecule, because it appears to greatly reduce background fluorescence relative in the fluorimeter.

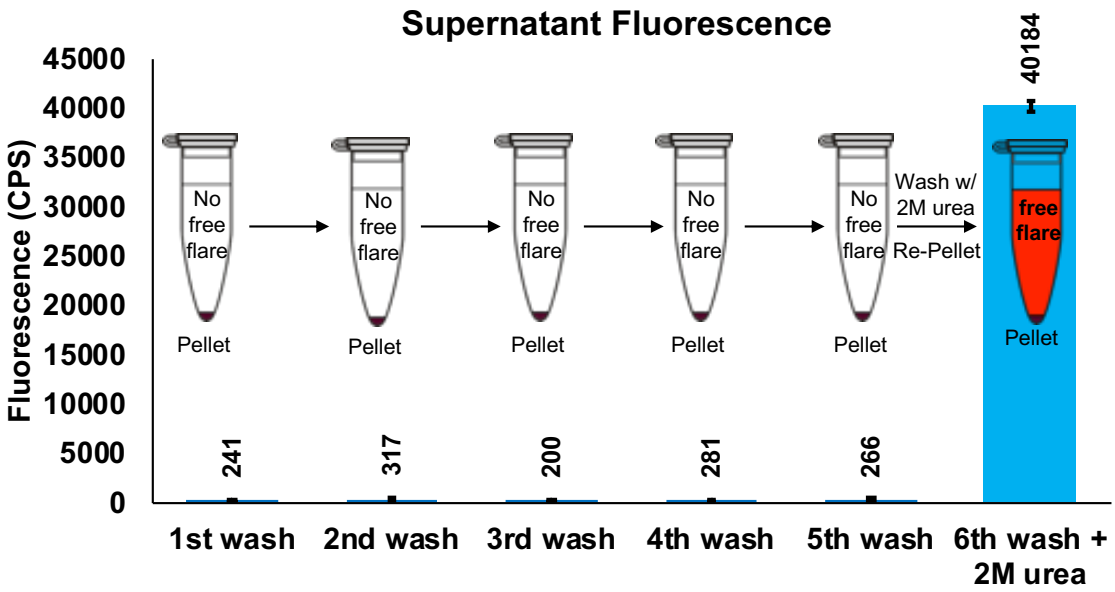


Figure 41. Pelleting experiment to qualitatively measure the aptamer-flare dissociation rate.

Second, in an induced fit system the stability of the flare-aptamer duplex could affect how well the target

molecule is able to change the conformation of the aptamer to displace the flare; so optimization for the stablest possible flare strand many not be ideal.

In order to further explore the induced-fit regime of aptamer NanoFlare dynamics, it was necessary to build a kinetic computational model of the system. This model would incorporate the additional species (aptamer-flare-DHEA-S intermediate, or ADF) and rate constants (k_F^{Apt} and k_R^{Apt} for the DHEA-S binding and dissociation from ADF, k_F^{Hyb} and k_R^{Hyb} for flare binding and dissociation from ADF) required for induced fit. Using the PySB software package, we developed aptamer Nanoflare kinetic models of both conformational selection and induced fit dynamics (**Code C2**).

While we had previously determined the equilibrium dissociation constants for the aptamer-flare and aptamer-DHEA-S interactions, the rate constants for both interactions are unknown. To explore the effects of different rate constants on the conformational selection model, we plotted the response of 100 nM of simulated aptamer-flare duplex to 1 mM DHEA-S while varying k_F^{Hyb} and k_R^{Hyb} and holding k_R^{Hyb} / k_F^{Hyb} = K_D^{Hyb} constant at the experimentally determined value of ~10 nM (**Figure 42**). While modifying the rate constants had no effect on the theoretical final concentration of unbound flare strand, it had a large effect on how quickly that final concentration is achieved; and particularly for k_R^{Hyb} values lower than 0.00001 s^{-1}, the model predicted aptamer NanoFlares would take hours to days to reach equilibrium.

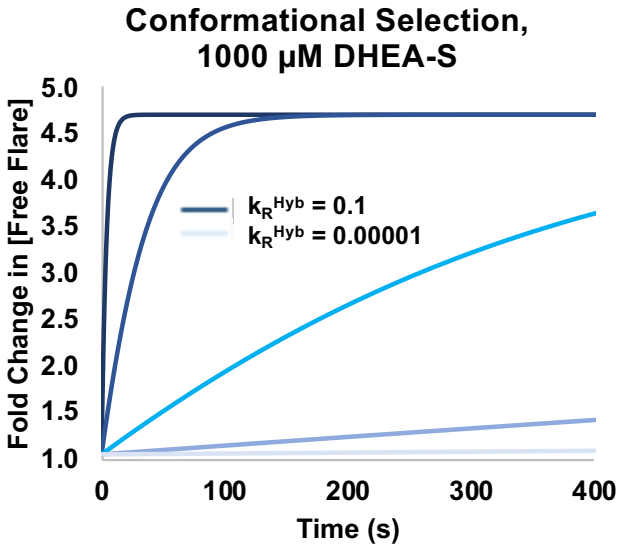


Figure 42. Conformational selection model's predicted response to 1000 μM DHEA-S, as a function of k_F^{Hyb} and k_R^{Hyb}. From darkest blue to lightest blue, k_R^{Hyb} values are 0.1, 0.01, 0.001, 0.0001, and 0.00001 s^{-1}. k_F^{Hyb} values are adjusted to keep k_F^{Hyb} and k_R^{Hyb} / k_F^{Hyb} = K_D^{Hyb} = 10 nM.

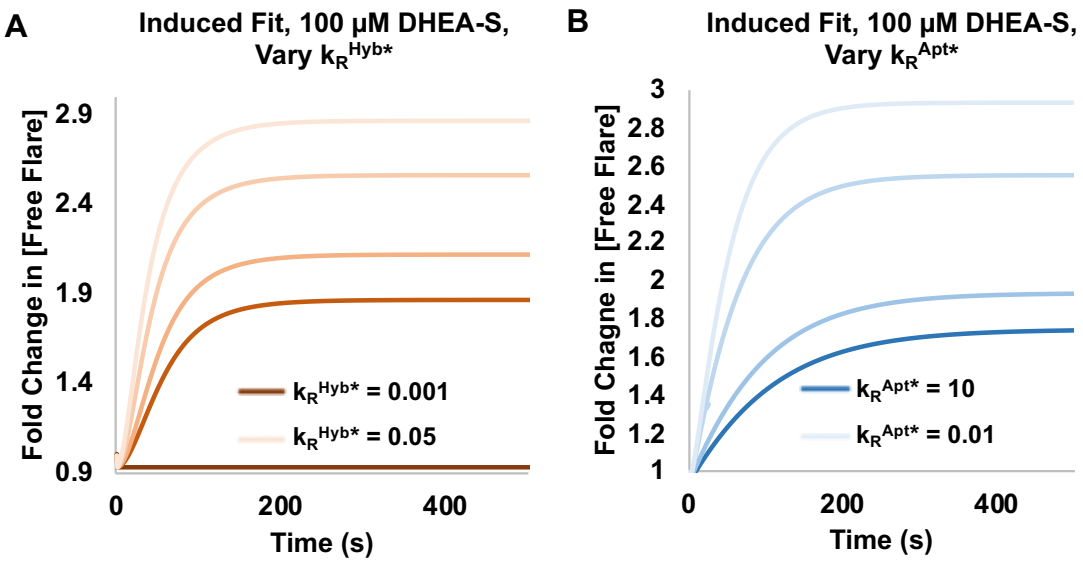


Figure 43. Induced fit model's response to 100 μM DHEA-S, as a function of **(A)** k_R^{Hyb*}, or **(B)** k_R^{Apt*}. For **(A),** k_F^{Hyb} stays constant at 0.1 $M^{-1}s^{-1}$, while k_R^{Hyb*}, from darkest to lightest orange, equals 0.001, 0.01, 0.015, 0.03, and 0.05 s^{-1}. For **(B)**, k_F^{Apt} stays constant at 0.001 $M^{-1}s^{-1}$, while k_R^{Hyb*}, from darkest to lightest blue, equals 10, 1, 0.1, and 0.01 s^{-1}.

We next explored how the new induced fit mathematical model behaved while varying its parameters. We plotted the response of 100 nM simulated aptamer-flare duplex to 100 μM DHEA-S while holding the aptamer-flare and aptamer-DHEA-S binding rates constant, and instead varying the dissociation rate of the flare strand (k_R^{Hyb*}) and DHEA-S (k_R^{Apt*}) from the ADF intermediate complex (**Figure 43**). The higher the flare dissociation rate from the ADF complex, the larger the fluorescent response was to DHEA-S. This means for optimal sensor performance, the ADF intermediate should ideally destabilize the aptamer-flare interaction as much as possible. Second, the lower the rate of DHEA-S dissociation from the ADF complex, the larger the fluorescent response was to DHEA-S. This means for optimal sensor performance, the ADF intermediate should feature as stable an aptamer-DHEA-S interaction as possible.

These results suggested important design rules/trade-offs for aptamer Nanoflare optimization. Given that the dissociation rate of flares from the ADF intermediate is likely connected to the stability of the aptamer-flare duplex, making a flare strand bind too tightly to the aptamer could actually decrease Nanoflare sensitivity to DHEA-S. Moreover, given that DHEA-S dissociation rate from the ADF

intermediate is inversely related to Nanoflare response to target, it could be important to design flare strands to bind to aptamers in regions that minimize disruption of the aptamer's target molecule binding pocket.

3.13 Measuring aptamer-flare-DHEA-S binding kinetics with bio-layer interferometry

Since the kinetic models for describing aptamer-Nanoflares require the binding and dissociation rates of the aptamer-flare and aptamer-DHEA-S interactions as parameters, measuring these rate constants is necessary to test and refine the accuracy of the models. We therefore sought to determine the aptamer-flare association rate (k_F^{Hyb}) and dissociation rate (k_R^{Hyb}) using bio-layer interferometry, or BLI (**Figure 44**). In the BLI experiments, biotinylated flare oligonucleotides were immobilized on the tip of single-use streptavidin-coated fiber optic probes. After equilibration in PBS buffer, each probe was incubated in a solution of aptamer, and the aptamer-flare binding was measured by the shift in the wavelength of light reflected from the tip of the BLI probe. Transferring the probe back to an aptamer-free buffer solution then allowed measurement of the aptamer-flare dissociation rate. Measuring the association and dissociation curves across a range of aptamer concentrations enables the determination of the aptamer-flare association rate (k_F^{Hyb}) and dissociation rate (k_R^{Hyb}). Moreover, it is possible to test the effect of DHEA-S on aptamer-flare dynamics by comparing aptamer-flare binding and dissociation rates in the presence and absence of DHEA-S.

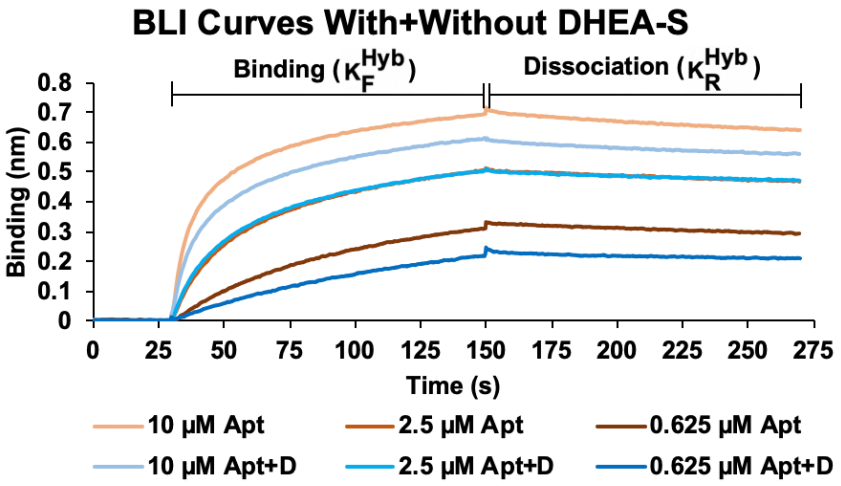


Figure 44. Measuring aptamer-flare hybridization with bio-layer interferometry (BLI). Representative BLI association and dissociation curves across a range of aptamer (Apt) concentrations, in the presence (blue) and absence (orange) of DHEA-S (D). For the blue curves, [D] = 300 µM.

One potential limitation of BLI is that modifying the flare oligonucleotide with biotin and immobilizing it on a streptavidin-coated surface may perturb its ability to bind to the aptamer. We therefore synthesized and screened three biotinylated flare variants (**Figure 45**): a 3'biotin flare, in which a biotin phosphoramidite was added directly to the 3' end of the flare sequence; a 3'biotin-$Sp18_2$ flare, which added two flexible spacer-18 (hexa(ethylene glycol)) phosphoramidites between the flare sequence and the 3' biotin; and a 5'biotin-$Sp18_2$ flare, in which the biotin and spacer-18 phosphoramidites were conjugated to the 5' end of the flare. The aptamer binding capacity of these flare variants was tested by fitting BLI binding and dissociation curves of each variant in the presence of 5 μM, 1.35 μM, and 450 nM aptamer (**Figure 45B**). The 3'biotin flare had the weakest binding (highest value) at K_D^{Hyb} = 481 nM. This was almost 50-fold weaker than the K_D previously estimated from melting curves (11 nM). We hypothesized that steric hindrance due to the flare's proximity to the streptavidin inhibited aptamer binding. This hypothesis was supported by the fact that the 3'biotin-$Sp18_2$ flare had a higher k_F^{Hyb} and lower k_R^{Hyb} than the 3'biotin flare, resulting in a 9-fold lower K_D^{Hyb} of 54 nM. This is only 5-fold weaker than the previously estimated K_D^{Hyb}. We trieated this value as a lower bound for aptamer-flare binding strength. Somewhat surprisingly, the 5'biotin-$Sp18_2$ flare displayed considerably weaker binding (K_D^{Hyb} = 177 nM) than its 3'biotin-$Sp18_2$ counterpart. This weaker binding may have been due to differences in the stability or accessibility of different parts of the aptamer's hairpin duplex, which made it easier for complementary oligonucleotides with an unbound 5' end (like the 3'biotin-$Sp18_2$ flare) to invade and displace the hairpin.

A

Probe Name	DNA Sequence
3'Biotin Flare	5'-GGA CAA CTT CGT-Biotin-3'
3'Biotin-$Sp18_2$ Flare	5'-GGA CAA CTT CGT-Sp18-Sp18-Biotin-3'
5'Biotin-$Sp18_2$ Flare	5'-Biotin-Sp18-Sp18-GGA CAA CTT CGT-3'

B

Probe Name	K_D, nM	k_F^{Hyb}, M/s * 10^4	k_R^{Hyb}, s^{-1} * 10^{-3}
3'Biotin Flare	**481**	**1.8** ± 0.1	**8.7** ± 0.3
3'Biotin-$Sp18_2$ Flare	**54**	**9.8** ± 0.2	**5.3** ± 0.6
5'Biotin-$Sp18_2$ Flare	**177**	**4.3** ± 0.1	**7.6** ± 0.2

Figure 45: Screening biotinylated flare probes for BLI experiments. (A) Sequences of the biotinylated flare probes that were synthesized and tested. Sp18 = spacer 18 hexa(ethylene glycol) phosphoramidite. (B) Aptamer-flare binding parameters measured with different biotinylated flare probes. Parameters calculated from fitting BLI curves of 5, 1.35, and .45 μM aptamer.

Due to the strong binding for the 3'biotin-$Sp18_2$ flare variant, we used this architecture for subsequent BLI measurements.

3.14 Effect of flare sequence and target molecule concentration on aptamer-flare binding kinetics.

The effect of flare length and structure on aptamer binding kinetics was investigated (**Figure 46**). Four truncated and biotinylated variants of the DIS11th_3T 12bp flare (12bpF) were synthesized: two 10 base pair variants, missing two nucleotides from either the 3' end (10bpF v1) or 5' end (10bpF v2) of 12bpF; and two 8 base pair variants, missing four nucleotides from either the 3' end (8bpF v1) or the 5' end (8bpF v2) of 12bpF. For each flare, BLI curves were generated for binding to the aptamer at a range of concentrations: 10, 5, 2.5, and 1.25 μM aptamer (12bpF, 10bpFv1); 20, 10, 5, and 2.5 μM aptamer (10bpFv2, 8bpFv1); or 40 and 20 μM aptamer (8bpFv2). Higher aptamer concentrations were used for the flares with weaker binding, in order to generate measurable binding curves. As expected, the 12bpF flare has the lowest overall dissociation constant (K_D^{Hyb}). This is primarily due to its much lower dissociation rate, as the 10bpF v1 association rate is similar to that of 12bpF. Both association rates are higher than that of 10bpF v2, suggesting that the two bases at the 5' end of the flare strand are important for rapid binding to the aptamer. As these bases bind to the end of the aptamer's self-complementary hairpin, they may be required to rapidly invade and displace it. Flares with truncated 5' ends (10bpF v2 and 8bpF v2) display slower binding and more rapid dissociation

A

Probe Name	Sequence
12bpF	5'-GGACAACTTCGT-Sp18-Sp18-biotin-3'
10bpF v1	5'-GGACAACTTC-Sp18-Sp18-biotin-3'
10bpF v2	5'-ACAACTTCGT-Sp18-Sp18-biotin-3'
8bpF v1	5'-GGACAACT-Sp18-Sp18-biotin-3'
8bpF v2	5'-AACTTCGT-Sp18-Sp18-biotin-3'

B

Probe Name	K_D^{Hyb}, nM	k_F^{Hyb}, $M^{-1}s^{-1}$ *10^4	k_R^{Hyb}, s^{-1} *10^{-2}
12bpF	**46**	**1.18** ± 0.01	**0.054** ± 0.001
10bpF v1	**800**	**1.27** ± 0.04	**1.02** ± 0.03
10bpF v2	**4,700**	**0.41** ± 0.02	**1.92** ± 0.05
8bpF v1	**4,900**	**0.41** ± 0.01	**2.01** ± 0.03
8bpF v2	**90,000,000**	**0.00** ± 0.07	**14** ± 2

Figure 46: Effect of flare length on aptamer-flare binding kinetics. (A) Sequences of the biotinylated flare probes that were synthesized and tested. Sp18 = spacer 18 hexa(ethylene glycol) phosphoramidite. **(B)** Aptamer-flare binding parameters measured with different biotinylated flare probes.

than flares of the same length with truncated 3' ends (10bpF v1 and 8bpF v1); in fact, binding is almost unmeasurably small for 8bpF v2. This is probably partly due to the role of the 5' nucleotides in aptamer hairpin displacement. Additionally, because most of the 10bpF v2 and 8bpF v2 sequences span a smaller hairpin in the aptamer, they are predicted to fold into hairpins which may disfavor hybridization to a complementary strand. These experiments suggest that flares should be designed to bind at the terminus of aptamer secondary structure, and that flare sequences that span hairpins should be avoided if possible.

We next sought to determine how flare length and location affect the response of aptamer-flare binding to the presence of DHEA-S (**Figure 47**). Aptamer binding to and dissociation from 12bpF, 10bpF v1 and 10bpF v2 was measured with 0, 30, 100, 300, and 1000 μM DHEA-S in the buffer. For all the flares, the more DHEA-S there was in the buffer, the lower the measured aptamer-flare association rate was.

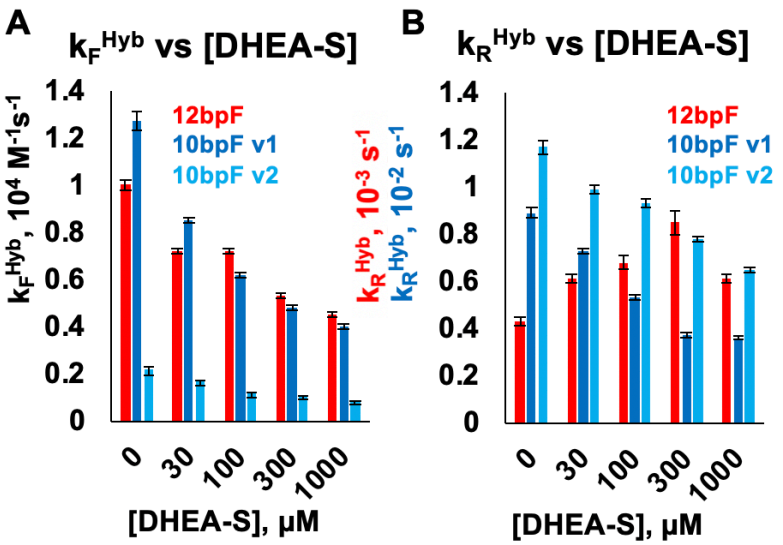


Figure 47: Effect of DHEA-S concentration on 12bp and 10bp flare-aptamer binding kinetics. (A) Observed k_F^{Hyb} as a function of DHEA-S concentration. **(B)** Observed k_R^{Hyb} as a function of DHEA-S concentration (red scale bar applies to 12bpF; blue scale bar applies to 10bpF v1 and 10bpF v2).

This indicated that DHEA-S competed with all the flares for binding to free aptamer. However, the dissociation rate for 12bpF responded to DHEA-S differently than either 10bpF v1 and 10bpF v2. For the tighter-binding 12bpF, adding DHEA-S slightly increases the dissociation rate, consistent with an induced-fit model of aptamer NanoFlare dynamics. Notably, the 12bpF dissociation rate decreased at the highest DHEA-S concentration; one possible explanation for this behavior is that at high concentrations the relatively hydrophobic DHEA-S may have formed transient micelles, which changed the way it interacted with the aptamer-flare duplex. Surprisingly, and unlike for 12bpF, the aptamer-flare dissociation rate for

both 10bpF v1 and v2 decreased when DHEA-S is added—in other words, the presence of DHEA-S appeared to stabilize, rather than destabilize, the aptamer-flare duplex. This counterintuitive effect replicated in multiple experiments performed on different days.

We sought to determine whether the surprising stabilizing effect of DHEA-S on 10bpF-aptamer binding was an aptamer-specific phenomenon, and to more generally measure the effect of aptamer structure on flare binding kinetics, by comparing aptamer-flare binding kinetics and DHEA-S response to a control sequence that contained the flare binding site but lacked the rest of the aptamer's structure (**Figure 48**). A control sequence (ctrl) was synthesized, which contained the 12bp region of the aptamer complementary to the flare strand, but replaced the rest of the sequence with adenosine nucleotides; this sequence should be able to bind to flare strands, but not to fold into the aptamer's secondary or tertiary structure. The association and dissociation rates of 10bpF v1 flare binding to both the aptamer and the control were measured in 0, 100, and 1000 μM DHEA-S. In the absence of DHEA-S, 10bpF v1 both bound and dissociated more quickly from the control sequence, suggesting that aptamer structure both inhibited flares from binding to free aptamers, and somehow stabilized already hybridized aptamer-flare duplexes. The effects of DHEA-S on aptamer-10bpF v1 binding and dissociation replicated: DHEA-S again appeared to compete for unbound aptamer (reducing observed k_F^{Hyb}) and stabilize aptamer-flare duplexes (reducing observed k_R^{Hyb}). There was no observable trend in the

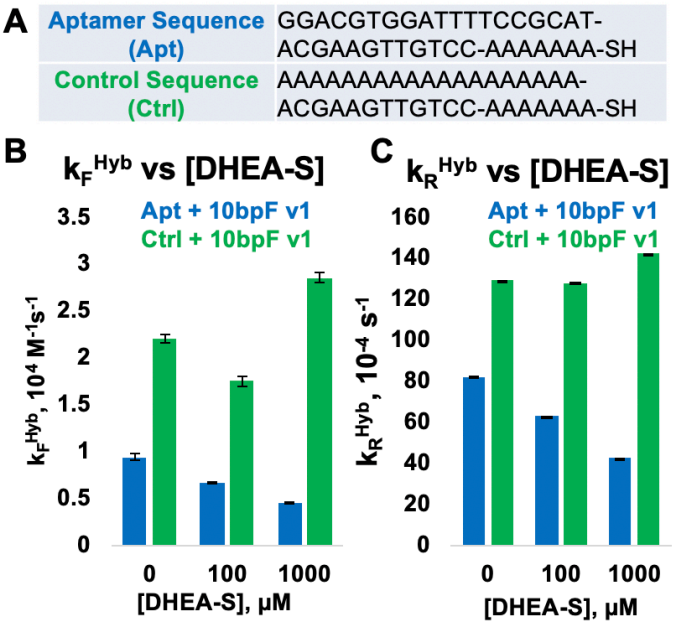


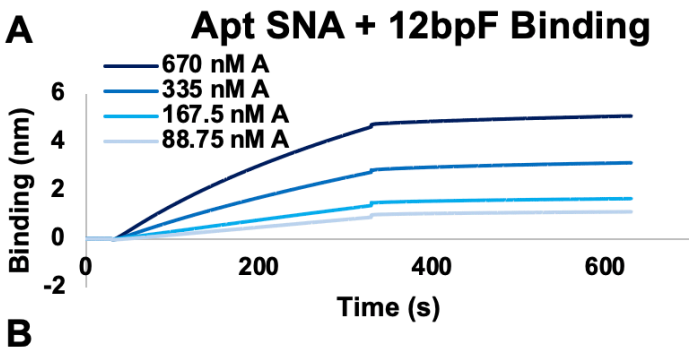
Figure 48: Effect of aptamer structure on flare binding kinetics and DHEA-S response. (A) Sequences of the DHEA-S binding aptamer (Apt), and a control sequence (Ctrl) which contains the flare-binding region of the aptamer but replaces all other bases with adenine. k_F^{Hyb} **(B)** and k_R^{Hyb} **(C)** were measured as a function of DHEA-S concentration, for 10bpF v1 binding to the Apt (blue) and Ctrl (green) sequences.

effect of DHEA-S on Ctrl-10bpF v1 association; and DHEA-S to had little to no effect on the dissociation rate of Ctrl-10bpF v1 duplexes. This suggested that the duplex-stabilizing effect of DHEA-S on aptamer-10bpF interactions was specific to and dependent on the structure of the aptamer.

3.15 Suitability of SNAs for bio-layer interferometry experiments

We tried using BLI to measure the kinetics of aptamer SNA binding to the flare strand, in order to see how they differ from free aptamer in solution. (**Figure 49**). BLI curves of four concentrations of aptamer SNAs binding to 12bpF flare were measured and fitted. This experiment was performed on SNAs with loading densities of 10, 31, and 55 aptamers per particle. The aptamer SNA-flare binding was specific: no binding is observed when aptamer SNAs were incubated with bare BLI probes in the absence of immobilized flare strands (data not shown). The aptamer-flare association rate for SNAs was roughly an order of magnitude lower than for free aptamers, though slightly higher for the SNA with the lowest loading density; this may reflect the relatively lower diffusion rate of the nanoparticles. Consistently, the dissociation rate of the SNA-flare interaction was so small the BLI instrument could not measure it. In fact, for some binding curves, the binding signal actually increased slightly during the dissociation step, when there are no SNAs free in the solution. Unmeasurably small dissociation rates were also observed in individual BLI curves of SNA binding to 10bpF v1, 10bpF v2, 8bpF v1, and 8bpF v2 (not shown). Moreover, addition of 1 M urea in the dissociation buffer failed to induce SNA-12bpF dissociation, though it increases the flare dissociation rate from free aptamer (not shown). We hypothesized that the source of this anomalously low dissociation

A



B

Apt/SNA	K_D^{Hyb}, nM	k_F^{Hyb}, $M^{-1}s^{-1}$ *10^4	k_R^{Hyb}, s^{-1} *10^{-4}
10	<0.001	0.32 ± 0.02	<0.001
31	<0.001	0.12 ± 0.02	<0.001
55	<0.001	0.14 ± 0.01	<0.001

Figure 49: Aptamer SNA-flare binding kinetics. (A) BLI curves of aptamer SNA binding to 12bpF flare. A = aptamer. **(B)** Kinetic parameters of aptamer-flare binding, for different SNA loading densities. Parameters derived from fitting BLI curves of 4 different SNA concentrations.

rate was polyvalent aptamer-flare interactions: since each SNA presented multiple aptamers on its surface, a single SNA could bind to multiple immobilized flare strands; and dissociation therefore required the simultaneous dehybridization of multiple independent aptamer-flare duplexes, a very rare event. These experiments suggest that kinetic measurement techniques that rely on flare immobilization to a surface may not be able to yield meaningful dissociation data for aptamer Nanoflares; kinetic measurements of fluorescence may be a more tractable way to measure these parameters in the future.

3.16 Cortisol aptamer-flare binding kinetics

We investigated whether the kinetic behavior of one aptamer-flare pair could predict the behavior of a different aptamer-flare pair with a similar design architecture, by designing aptamer-flare pairs for the cortisol-binding aptamer Cor10F (**Figure 50**). Chavez and Mirau labs had previously shown that Cor10F had a cortisol dissociation constant of 750 nM; and mFold structural prediction

A

Name	Target	Sequence
DIS11th_3t	DHEA-S	GGACGTGGATTTTCCGCATA-CGAAGTTGTCCA
Cor10F	Cortisol	GGACGACGCCAGAAGTTTACG AGGATATGG-TAACATAGTCGT

B

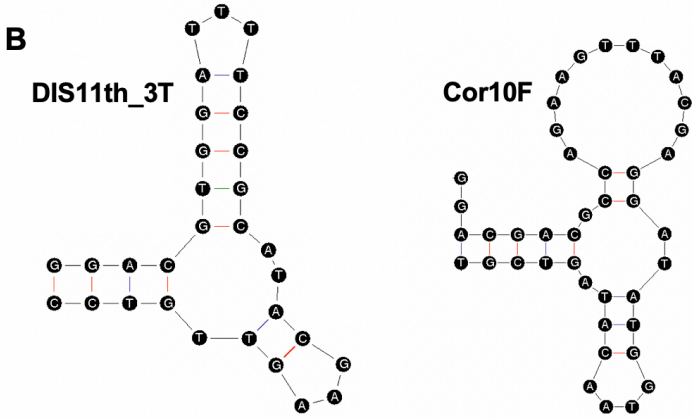


Figure 50: Design of cortisol-binding aptamer-flare pair. (A) Name, target, and sequence of the DHEA-S binding aptamer we have investigated (DIS11th_3T), and of a cortisol-binding aptamer (Cor10F). Flare-binding aptamer sequence in red. **(B)** mFold structure predictions of DIS11th_3T and Cor10F.

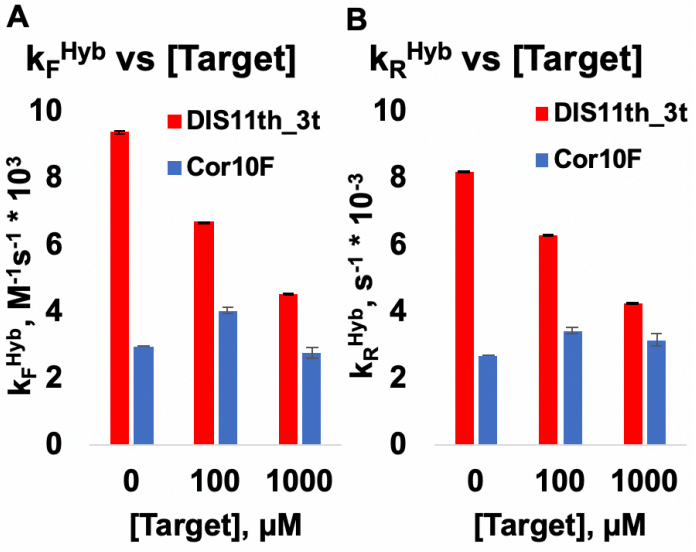


Figure 51: Response of (A) aptamer-flare association rate (k_F^{Hyb}) and (B) aptamer-flare dissociation rate (k_R^{Hyb}) to the presence of target molecule, for DHEA-S and cortisol binding aptamers. DHEA-S binding aptamer DIS11th_3t in red, cortisol binding aptamer Cor10F in blue. Target molecule added is DHEA-S for DIS11th_3T and cortisol for Cor10F.

suggests that Cor10F has a 3-stem secondary structure similar to that of DIS11th_3T (**Figure 50B**). Cor10F was synthesized, along with 10 bp, 3' biotinylated flare sequences designed to hybridize to the aptamer's 3' end, similar to the DIS11th_3T flares we had already investigated. Then, the kinetics of aptamer-flare binding and dissociation were measured in the presence and absence of the aptamer's target molecule, using BLI (**Figure 51**). The Cor10F aptamer-flare pair displayed lower association (k_{Hyb}^{F}) and dissociation (k_{Hyb}^{R}) rates than the DIS11th_3t aptamer-flare pair, possibly due to the longer Cor10F terminal hairpin and lack of flare secondary structure, respectively. While increasing concentrations of DHEA-S led to lower observed k_{Hyb}^{F} and k_{Hyb}^{R} for the 10 bp flares of DIS11th_3T, no similar trend was observed for Cor10F in the presence of increasing concentrations of cortisol. These results suggested, narrowly, that this particular Cor10F aptamer-flare pair did not respond structurally to the presence of its target molecule; and generally reinforced the idea that in the absence of modeling and design rules that better represent aptamer-flare-target dynamics, more comprehensive exploration of aptamer-flare design space was required.

3.17 Microarray screens of aptamer-flare pairs

In order to more comprehensively explore aptamer-flare design space, we designed and performed microarray screens for stable aptamer-flare duplexes that respond to DHEA-S (**Figure 52**). Chavez lab first compiled a list of known DHEA-S binding aptamers, as well as aptamers that bind to other target molecules

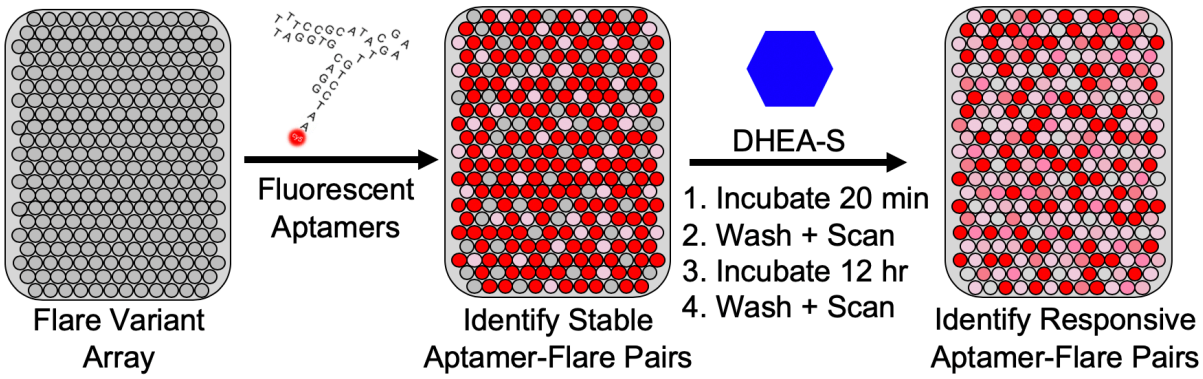


Figure 52. Microarray screen for stable and DHEA-S responsive aptamer-flare pairs. An array of 8-12 bp flare variants is incubated and washed for 30 minutes with Cy3- and Cy5-labeled aptamers and scanned to identify stable aptamer-flare pairs. In the first round of experiments, the array was incubated with DHEA-S for 20 minutes, washed and scanned, and then incubated again with DHEA-S for 12 hours before a final washing and scanning, with the goal of identifying aptamer-flare pairs that dissociate in the presence of the target molecule.

of interest like cortisol, dopamine, and TNT (**Table A2**). Then, for each aptamer, a library of every possible 8 bp, 9 bp, 10 bp, 11 bp and 12 bp complementary flare sequence was generated. 7 replicate spots of each flare sequence were arranged in a random position of a 60,000 feature array, and then 8 identical copies of that array were synthesized in 8 positions (wells) on Agilent slides. Cy3-modified DNA or Cy5-modified RNA aptamers were then hybridized onto the arrays, and the arrays were scanned to measure aptamer-flare hybridization efficiency. In a first set of experiments, three subarrays were then incubated with 30 μM DHEA-S, three subarrays were incubated with 300 μM DHEA-S, and two control wells were incubated with PBS buffer. In a second round of microarray experiments, subarrays were incubated for 300 μM DHEA-S and cortisol in both PBS and SELEX buffer 1 hour at 25ºC with. And in a third round of experiments, subarrays were incubated with 300 μM of various other target molecules for 1 hour at 25ºC, including TNT, riboflavin, dopamine and theophylline.

A computational analysis pipeline was developed for processing the microarray data using Jupyter Notebooks and Python (**Code C3**). The fluorescence intensity values of every array feature from each step of a microarray experiment (roughly 2 million data points) was consolidated into a Pandas DataFrame and labeled with relevant information for analysis, such as the flare sequence, the name and sequence of the aptamer the flare binds to, the flare length and binding position along the 5'-3' sequence of the aptamer, the sub-array within which the feature is located, and experimental step and conditions of the sub-array the feature occupies. Then, the data set was cleaned to remove all array features flagged by the Agilent software as having high background, nonuniform signal across the feature spot, or whose fluorescence intensity was an outlier relative to the other replicate spots with the same sequence in that sub-array. Then the mean, median, and standard deviation of the fluorescence intensities of the remaining replicate spots were calculated in each sub-array for each experimental step. The mean fluorescence intensity thus calculated was used as the basis for subsequent analysis.

Having cleaned, labeled, and processed the raw array fluorescence data, we first sought to determine the effect of flare length and flare binding position on aptamer-flare hybridization efficiency (**Figure 53**). This analysis initially focused on the DIS11th_3 aptamer, because the aptamer we have been using to develop and test aptamer Nanoflares for the past year is a truncated version of DIS11th_3. The dataset was filtered to include only data from the initial hybridization step of the experiment. Then, the data for all the DIS11th_3 flare strands across all the sub-arrays were grouped by either flare length, or by the position of the 5' end of the flare-binding region of the aptamer along the aptamer sequence, and the average and standard deviation of each of these groupings was plotted. Average aptamer-flare hybridization increased as flare length increased, as would be expected (**Figure 53A**). However, the large standard deviations indicated that for any given flare length, there was a lot of variation in hybridization efficiency. Plotting hybridization efficiency as a function of flare binding position revealed the source of some of this variation (**Figure 53B**): flare binding position strongly affected hybridization efficiency. In the case of the 12bp flares for DIS11th_3, flares binding at the 5' and 3' ends of the aptamer hybridized efficiently, as did those that start binding in a region 11-14 bases inside the aptamer sequence, while a region 20-25 bases along the

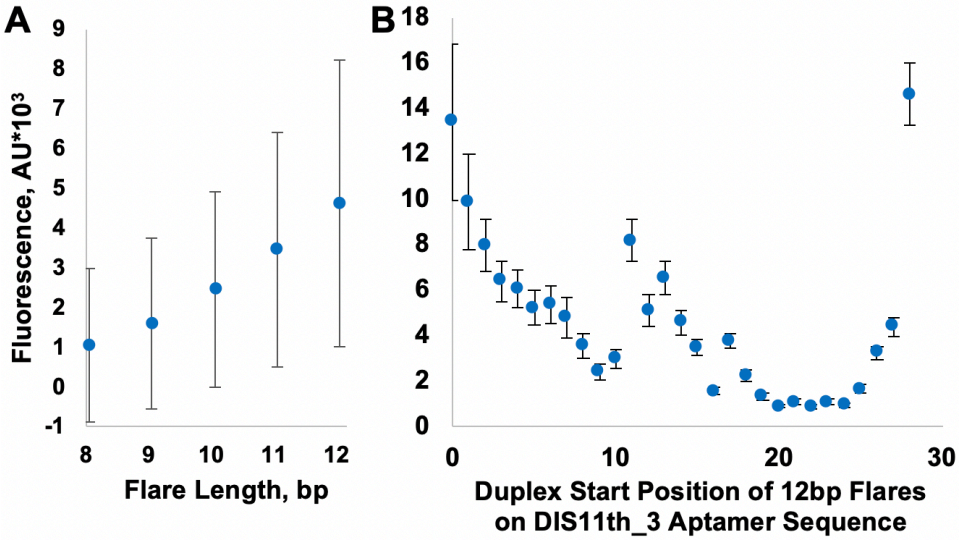


Figure 53. Effect of flare length and binding position on hybridization efficiency to DIS11th_3 aptamer. **(A)** Flare hybridization efficiency (as measured by mean fluorescence on the array) as a function of flare length. **(B)** Flare hybridization efficiency as a function of the start position of the flare binding region on the 5'—3' aptamer sequence, for 12bp flares.

aptamer sequence barely hybridized at all. We hypothesized that the middle region that hybridized well represented an area with more open or accessible secondary structure, while the region that did not hybridize well was part of a stable hairpin.

We next sought to determine if there were any consistent patterns or design rules for well-hybridizing flares across all the aptamers we tested. Toward this end, heat maps of aptamer-flare hybridization efficiency as a function of flare length and binding position were generated with MatPlotLib for every aptamer tested (two representative heatmaps for aptamers Dopa2 and 10_51 are show in **Figure 54**).

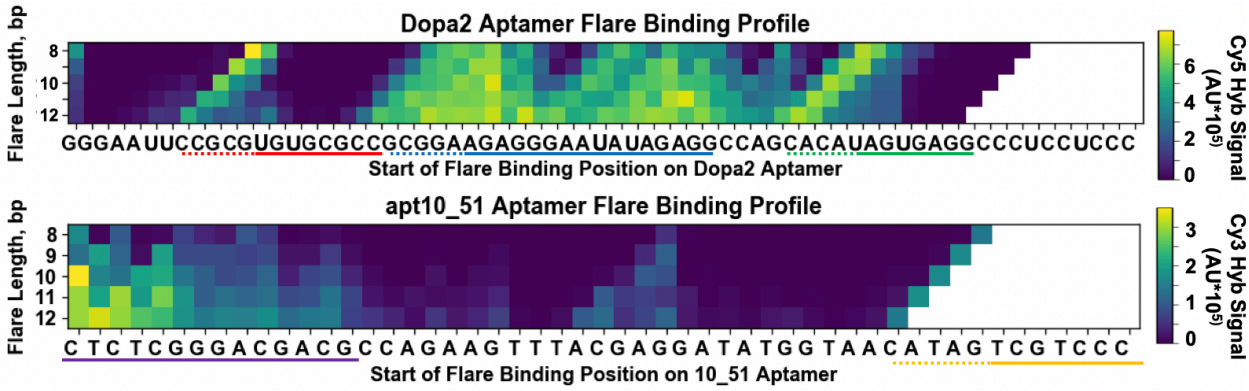


Figure 54: Initial aptamer-flare hybridization heatmaps for the Dopa2 dopamine-binding RNA aptamer, and the 10_51 DHEA-S binding DNA aptamer. Colored lines represent regions of sequence within which flares can stably bind. Solid lines represent core regions required for flares of any length to bind.

The flare binding profile of 10_51 is consistent with what was observed DIS11th_3: the 3' and particularly the 5' end of the aptamer show strong flare binding. By contrast, the Dopa2 aptamer shows little flare hybridization at its 3' or 5' ends, but strong hybridization at particular internal sites. The heatmaps plotted hybridization intensity as a function of the length of the flare strand, and of the location along the aptamer sequence where the 3' end of the flare strand starts to bind. Both the Dopa2 and the 10_51 heatmap showed diagonal features (underlined by the dotted red, blue, green and yellow lines in Figure 4) which suggested that the 5' end of flare strands initiate binding at particular bases on the aptamer (such as the 3' end of the aptamer in the case of 10_51). We hypothesized that labile secondary and tertiary aptamer

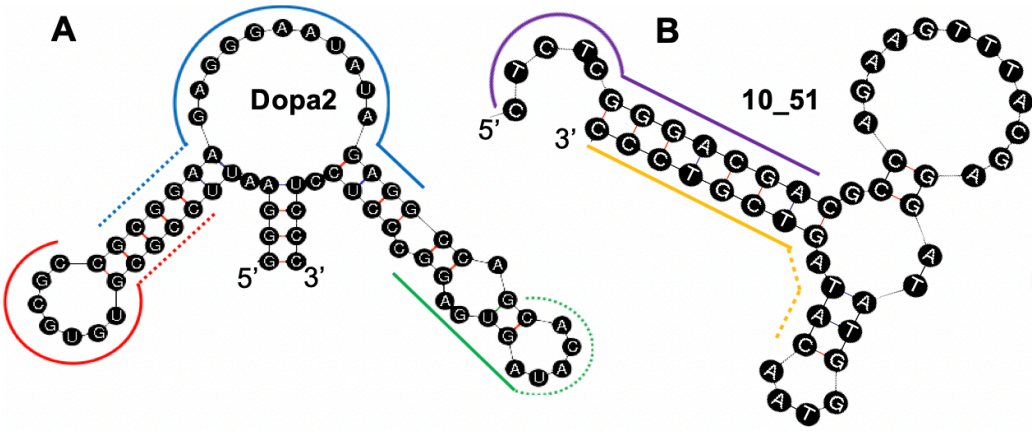


Figure 55: Structural analysis of flare-binding aptamer regions for the (A) Dopa2 and (B) 10_51 aptamers with mFold. Colored lines represent regions of sequence within which flares can stably bind, corresponding to Figure 4. Solid lines represent core regions required for flares of any length to bind.

structure susceptible to strand invasion by the 5' end of the flare strand is the source of these diagonal features on the heatmaps.

To further understand the structural basis of efficient aptamer-flare binding, we compared mFold secondary structure predictions for the aptamers to the heatmap regions that show strong flare binding (Dopa2 and 10_51 shown as examples in **Figure 55**). Consistent with our hypothesis that strong flare binding correlates with labile secondary structure, flares tended to bind to Dopa2 where multiple unpaired bases were predicted, such as in loops or at terminal overhangs. For 10_51, the most stable aptamer-flare pairs were formed at a predicted unpaired overhang on the 5' end of the aptamer. Importantly, not every predicted loop region hybridized to flares on the microarray, and the 5' and 3' aptamer ends didn't always generate stable flare pairs either; but it was generally the case across multiple aptamers that stably hybridizing flares cluster in these regions of sequence. Further work and more detailed modeling may reveal which loops and termini are more or less suitable for flare binding.

In addition to analyzing the structural determinants of aptamer-flare binding, we also explored the ability of a thermodynamic model of to predict hybridization efficiency. In collaboration with Chavez lab at AFRL, UNAFold software was used to generate predictions of the ΔG of aptamer-flare hybridization (ΔG_{Hyb}) in 1X PBS at 25ºC, for every flare on the microarray. The relationship between experimental hybridization and both flare length and ΔG_{Hyb} was then plotted (**Figure 56**). Consistent with previous microarray experiments, flare length alone is a poor predictor of hybridization efficiency: while there is a weak overall trend of higher hybridization fluorescence with longer flares, a large proportion of flares of every length completely fail to hybridize to the complementary aptamer. Even the shortest 8 bp flares contained a significant population that hybridized to the aptamer. By contrast, UNAFold's calculated ΔG_{Hyb} values showed more predictive ability than flare length, especially at the margins: no flare with a predicted $\Delta G_{Hyb} > -10$ kcal/mol hybridized to the aptamer, while every flare with a predicted $\Delta G_{Hyb} < -16$ kcal/mol hybridized to at least some extent with the aptamer. These guidelines will be useful for designing future flare constructs, enabling the exclusion of flare designs that won't work, and also potentially guiding designers toward shorter flare strands that can stably bind an aptamer while disrupting a smaller portion of its secondary and tertiary structure.

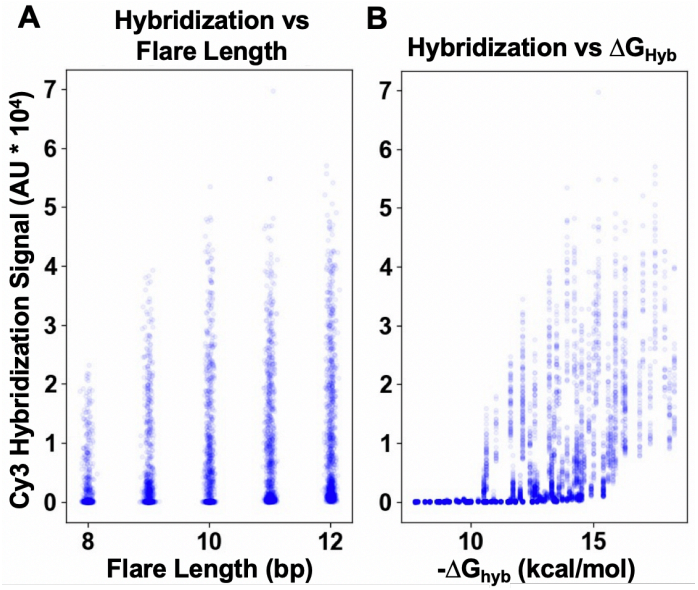


Figure 56: Relationship between hybridization efficiency, flare length, and predicted ΔG of flare hybridization (ΔG_{Hyb}), for the aptamer DCA6th_23. (A) Flare hybridization versus flare length. **(B)** Flare hybridization versus $-\Delta G_{Hyb}$, as calculated by UNAFold.

3.18 Towards discovery of target-responsive aptamer-flare pairs

We analyzed the response of all the tested aptamer-flare pairs to incubation with DHEA-S (**Code C4**). Relative loss of hybridization for each flare sequence in each sub-array was calculated by dividing the

fluorescence after incubation with DHEA-S by the fluorescence measured in the previous experimental step. Then, for the three sub-arrays treated with 300 μM DHEA-S, the change in fluorescence of each flare sequence was averaged; and the same was done for the three sub-arrays treated with 30 μM DHEA-S. To correct for loss of fluorescence due to spontaneous aptamer dehybridization during washing and incubation steps, and due to bleaching of the Cy3 fluorescence, the change in hybridization was calculated for one of the two the sub-arrays treated only with PBS (the other showed anomalously high fluorescence at all array features during one of the experimental steps, and was excluded from analysis). Then, the averaged relative changes in hybridization fluorescence for the wells treated with DHEA-S were normalized to the change in fluorescence observed for each flare sequence in the PBS-only well. In this way, a 'Hybridization Signal' value of 1 means that the aptamer-flare pair lost exactly as much fluorescence as the PBS control; a value >1 means the pair lost less fluorescence than the control; and a signal <1 means the pair lost more fluorescence than the control, which is what would indicate a response to DHEA-S. To minimize artifacts due to low hybridization efficiency, we excluded from analysis all the data from all flare sequences with initial hybridization signals less than 5 standard deviations higher than the background fluorescence signal of nonbinding features on the array.

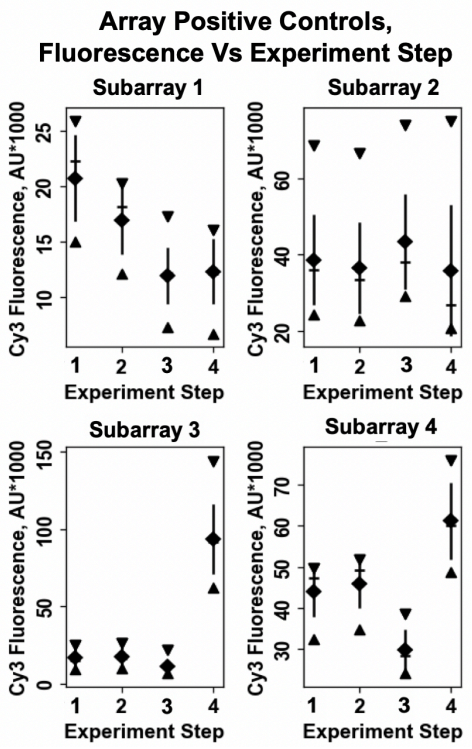


Figure 56: Variation in fluorescence of the microarray's positive controls between subarrays and experimental steps.

◆ = mean, – = median, | = standard deviation, ▼ = maximum, ▲ = minimum.

Despite these measures, we were unable to identify aptamer-flare pairs that consistently, across replicate subarrays, dehybridized relative to controls in the presence of the aptamer's target molecule. Particularly confounding was the observation that measured hybridization fluorescence actually increased

after incubation for some aptamer-flare pairs, which seemed likely to be the result of an error in the instrument's measurement. Each subarray contained 12 replicate Bright Spot positive controls, which were array features covalently modified with Cy3 fluorophores. To better understand the sources of noise in our experimental system, we calculated and plotted statistics on these array positive controls within each subarray, and for each step in the experiment: (1) hybridization scan 1, (2) hybridization scan 2, (3) incubation 1 scan, and (4) incubation 2 scan (**Figure 56**). The measured fluorescence intensity of the array positive controls varies both between subarrays and between experimental steps, in some cases (like step 4 in subarray 3) by a great deal.

These measurements are highly useful for designing array experiments and analysis going forward. For one thing, it is clear that absolute fluorescence/hybridization values cannot be compared between subarrays; normalization relative to a previous experimental step is required. Moreover, this normalized change in fluorescence must be further corrected for variation in the intensity of the scanner's measurement—this might be achieved through normalizing to the array's bright spot controls, or (if measured fluorescence intensity varies across a single subarray) it may require the design of a larger number of fluorescent control features into future arrays. These results are encouraging, because they indicate that the difficulty thus far in identifying aptamer-flare pairs could be corrected by implementing these changes to the experimental design and analysis.

3.19 Materials and Methods

3.19.1 Materials. Unless otherwise noted, all reagents were purchased from commercial sources and used as received. Oligonucleotides are synthesized using phosphoramidites and associated reagents Glen Research, Co. (Sterling, VA, USA); or ordered from Integrated DNA Technologies. 13 nm Citrate capped gold nanoparticles were synthesized as previously described.[137] Microarray slides were purchased from Agilent. All other reagents were purchased from Sigma-Aldrich (St. Louis, MO, USA).

3.19.2 Aptamer NanoFlare Synthesis. Thiolated aptamer sequences were reduced by incubation with 10 mM DTT in 100 mM Tris buffer for 30 minutes. DTT was removed from the aptamer solution by size

exclusion on a GE NAP5 column. Flare strand is then added to the thiol-aptamer solution, which is then heated to 65ºC and slowly cooled to room temperature to anneal flare and aptamer. AuNPs were mixed with 0.2% Tween 20 detergent, and 5 M NaCl was added to a final concentration of 150 mM NaCl. AuNP solution is vortexed, and then thiolated aptamer is added to the AuNPs in a 400:1 aptamer:AuNP solution. The solution is vortexed for 20 seconds, sonicated, and incubated in a covered shaking container overnight. Then more NaCl was added to the solution from 5 M stock solution, bringing the final NaCl concentration to 0.35 M. 1000-fold excess of thiolated PEG oligomer was then added to the solution, which was then incubated for 6 hours. Excess oligonucleotides were removed by repeatedly pelleting the particles in a centrifuge (10-20,000 RCF), and then resuspending in fresh 1X PBS. DNA loading density was quantified by Oligreen assay.

3.19.3. Nuclear Magnetic Resonance of Aptamers. Lyophilized DIS11th_3 and DIS11th_3T were resuspended to a concentration of 0.1 M in 20 mM deuterated Tris, 50 mM NaCl, and 10 mM $MgCl_2$ at pH 7 in 90:10 H_2O:D_2O. For the aptamer with DHEA-S measurements, 1 mg/mL DHEA-S dissolved in MeOH was dried in a microcentrifuge tube centrivap, then resuspended in the aptamer solution to a 1.25:1 DHEA-S:aptamer ratio. Spectra were acquired on a 600 MHz Bruker spectrometer using the Watergate pulse sequence for water suppression.

3.19.4 Fluorescence measurements. Fluorescence of aptamer NanoFlares was measured in a Bio-Tek H4 plate reader, or in a Horiba Fluorolog-QM fluorimeter. Fluorimeter samples were prepared in 1 mL quartz sample cuvettes

3.19.5 Isothermal titration calorimetry. ITC experiments were performed on a Malvern MicroCal PEAQ-Automated instrument. Cell and syringe buffers were equilibrated by NAP column. Enthalpic curve fitting was performed using MicroCal ITC-Origin analysis software.

3.19.6 Bio-layer interferometry. BLI experiments were performed on a BLItz Bio-Layer Interferometer; binding curves were fitted and analyzed with BLItz Pro software. Before use, streptavidin-modified probe tips were hydrated in PBS for 10 minutes. Flare-modified probe tips were prepared by vortex-incubating

300 μL of 10 μM biotinylated flare sequences suspended in 1X PBS with streptavidin modified probe tips for 2 min, then vortex-incubating the tip in 1X PBS for 2 more minutes. 300 μL samples of 10, 5, 2.5, 1.25 and 0.625 μM aptamer in PBS were incubated with the flare modified probe tip for 2 minutes to measure association rate, and then the tip was vortex-incubated with buffer alone for 2, 5, or 10 minutes to measure dissociation rate. For samples testing DHEA-S response, the DHEA-S was added to both the aptamer/association solution, and to the buffer-only/dissociation solution.

3.19.7 Microarray screens. Each Agilent microarray slide was blocked against nonspecific binding by submerging in SuperBlock buffer in a gentle rotary shaker for 30 minutes. After blocking, a 2 μM mixture of Cy3-DNA and Cy5-RNA aptamers in 1XPBS + 0.05% Tween 20 was incubated in the wells for 30 minutes at 25ºC to form flare-aptamer duplexes. The arrays were then washed and scanned in the Cy3 and Cy5 channels with an Agilent SureScan Dx Microarray Scanner to measure flare hybridization. Incubation for 30 more minutes in fresh buffer at 25ºC followed by washing and scanning was performed to identify aptamer-flare pairs that hybridized stably to each other. The wells were then incubated for 20 minutes at 25ºC with the target molecule(s) in 1X PBS + 10 mM $MgCl_2$ + 5 mM KCl + 0.05% Tween 20 (hereafter PBSb), or in 20 mM HEPES + 1 M NaCl + 10 mM $MgCl_2$ + 5 mL KCl + 0.05% Tween 20 (hereafter SELEXb), which is the buffer many SELEX aptamer evolution experiments are conducted in. After collecting all the scans, Agilent analysis software was used to align the array images with maps of the array spot names and identities. To test for rapid aptamer-flare response to target molecule, the wells were then incubated for 20 minutes at 25ºC with DHEA-S in PBS + 0.05% Tween 20 before washing and scanning. For this experiment, three wells were incubated with 30 μM DHEA-S, three wells were incubated with 300 μM DHEA-S, and two control wells were incubated with PBS + Tween. To test for aptamer-flare pairs that respond more slowly to the presence of target molecule, the wells were incubated overnight (~12 hours) with the same concentrations of DHEA-S as before, and the slide was washed and scanned one last time. After collecting all the scans, Agilent analysis software was used to align the array images with maps of the array spot names and identities.

CHAPTER 4: Exploring the Limits of Cytosolic Enzyme Delivery with CRISPR SNAs

---

Collaborators included Jessica Rouge, Jeff Brodin, Resham Banga, Kevin Zhao, Robert Stawicki and Chad Mirkin

4.1 Introduction

CRISPRs are Clustered Regularly Interspersed Short Palindromic Repeats of DNA in the genomes of many bacteria and archaea.[163] In type II CRISPR systems, the operon containing these repeats produces two RNA molecules, the CRISPR RNA (crRNA) and the trans-activating crRNA (tracrRNA), which hybridize to form a constant hairpin tertiary structure with a 5' variable (or guide) region ~20 bases long.[164] In 2012, Doudna lab demonstrated that these two RNAs can be fused by a short hairpin to produce a single short guide RNA (sgRNA) without any loss of function.[165] This RNA structure binds to a large nuclease called CRISPR-Associated protein 9 (Cas9), forming a 200 kilodalton Cas9/sgRNA ribonucleoprotein complex (RNP) (**Figure 57**). The Cas9/sgRNA ribonucleoprotein (RNP) binds to and interrogates double stranded DNA for a short sequence motif, called Protospacer Adjacent Motif (PAM), which the Cas9 recognizes.[166] In the most extensively studied CRISPR/Cas9 system, derived from *Streptococcus pyogenes*, the SpCas9 nuclease recognizes an NGG PAM, where N is any nucleotide.[166] The Cas9/sgRNA complex cleaves DNA duplexes that are adjacent to a PAM and complementary to the sgRNA's 17-20 bp guide region.[167] Changing the guide region of the sgRNA programs the SpCas9 nuclease to cleave any ~20 bp DNA duplex that is followed by an NGG motif.[165] In 2013, Zhang lab demonstrated this programmable CRISPR endonuclease can edit loci in the genomes of mammalian cells.[168]

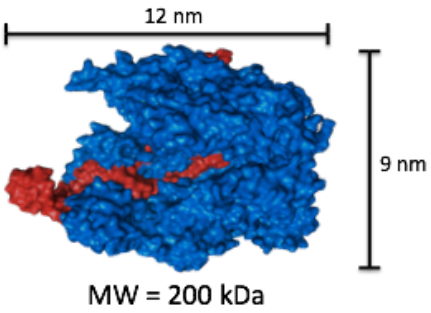


Figure 57. Cas9/sgRNA ribo-nucleoprotein complex. Cas9 (blue) binds sgRNA (red).

CRISPR is not the first tool developed for targeted genome editing of mammalian cells, but it has several advantages over its predecessors. Zinc finger nucleases (ZFNs) and transcription activator-like effector nucleases (TALENs) can also be designed to cleave specific DNA sequences.[169, 170] However, designing a zinc finger nuclease to target a new DNA sequence requires several months of protein engineering and costs thousands of dollars, because the protein sequence of ZFN does not dictate DNA sequence binding specificity in a predictable, programmable way.[171] TALENs are programmable: each

subunit binds one DNA base, and its sequence specificity is dictated by two amino acids.[172] However, a new TALEN gene must be synthesized for each new targeted sequence, and the repeated sequences from each subunit make such synthesis complex.[173] By contrast, reprogramming the Cas9 nuclease only requires changing the 20 bp at the 5' end of the sgRNA, a process simple enough that two sgRNAs can be simultaneously introduced into a gene editing plasmid with a single commercially purchased DNA oligonucleotide.[174]

SpCas9 was the first CRISPR nuclease to be successfully applied to genome editing in mammalian cells, but it is not the only one. Since 2013, myriad gene editing tools have been developed from the CRISPR nucleases of different species. *Staphylococcus aureus* Cas9 (SaCas9) is similar to spCas9, but has a different PAM and is 20% smaller, which allows the saCas9 gene to be more easily packaged into adeno-associated viral gene therapy vectors.[175] CRISPR protein from *Prevotella* and *Francisella* (Cpf1) nucleases from the class II CRISPR system differ significantly from SaCas9 or SpCas9: the native CRISPR-Cpf1 system uses a single crRNA which is half the size of the fused sgRNA of SaCas9 or SpCas9, recognizes an AT-rich PAM that enables it to target regions of genomes where the GC-rich PAMs of saCas9 or spCas9 are rare, and is more specific than Cas9, generating very few off-target cleavage events in edited cells or embryos.[176, 177] CRISPR systems have been further engineered through rational design and directed evolution to increase genome cleavage specificity, and to function with a broader range of PAM sequences.[178, 179]

Since 2012, CRISPR has been deployed in an explosion of precision genome editing applications in both basic biological research and therapeutics. Cleaving gene sequences with CRISPR frequently induces insertion/deletion (indel) mutations, which can shift the reading frame of a gene and disable its function. [168] One reason CRISPR is so powerful is that delivering multiple sgRNAs enables Cas9 to easily target and knock out multiple sites or genes simultaneously. Cas9 has been used to specifically and simultaneously knock out up to 62 genes in a single cell.[180] Alternatively, Cas9 delivered with a library of sgRNAs targeting every gene in a cell (called a Genome-scale Cas9 Knock Out library, or GeCKO) can be transduced into

cells to screen for genes responsible for relevant phenotypes, like synthetic lethal genes or genes which promote metastasis when knocked out in cancer cells.[181, 182]

In addition to destroying gene function via indel mutations, Cas9/sgRNA RNPs can also introduce highly specific gain-of-function edits or gene insertions. RNPs can be co-delivered with donor template DNA containing homology arms identical to the DNA sequence near the site targeted by the sgRNA. Cells can use the donor template DNA to repair the double strand break via homology directed repair.[183] Combining CRISPR with homology directed repair of a donor template therefore enables the one-step insertion of precise genetic edits.[184] In models like mice where such editing was previously difficult and time-intensive, CRISPR now enables easy tagging of genes with epitope tags or conditional knockout machinery, and the introduction at precise loci of large genetic constructs, such as an inducible Cas9 gene.[185] By fusing nuclease-null Cas9 (dCas9) genes to adenine or cytosine deaminase domains, it is even possible to directly interconvert specific DNA bases, introducing precise genomic edits without requiring a DNA template in a process now known as base editing.[186, 187, 188] By fusing dCas9 to a reverse transcriptase domain, it is even possible to use an extended version of the guide RNA as a template to introduce sequence-defined genetic insertions, deletions, and other conversions directly into a cell's genome without requiring donor template DNA or homologous recombination.[189]

CRISPR systems have also been modified to perform a number of functions besides genetic editing. Catalytically inactivated dCas9 has been fused to transcriptional activation and repression domains, thereby enabling programmable control of gene expression.[190, 191] The dCas9 transcriptional activator in particular enables novel screens analogous to siRNA or CRISPR knockout libraries, but where genes are over-expressed.[192] dCas9 fused to fluorescent proteins allows microscopic tracking of specific sites in the genome and the study of sequence-specific nuclear organization.[193] Finally, active Cas9 can be targeted to cleave a variety of nonfunctional genomic regions in a zygote, and the frequency and sequence of the mutations in each cell of the mature organism can be used to track lineages of cell differentiation during embryonic development.[194]

One of the most exciting applications for CRISPR and programmable nucleases is the prospect of correcting disease with a precise genetic knockout or edit. Previous gene therapies either randomly inserted or temporarily expressed genes to replace loss of function mutations.[195] CRISPR enables the correction of disease alleles at their native site in the genome, such as a loss of function in *Fah* causing tyrosinemia.[196] CRISPR also enables treatments that knock out genes. For instance, HIV requires the cell surface receptor *CCR5* to infect T cells, and using CRISPR *ex vivo* to knock out *CCR5* in hematopoietic cells renders humanized mice resistant to HIV infection.[197]

With the explosion in research and application of CRISPR and RNA-guided nucleases, the rate limiting challenge in gene editing has shifted from experimental design to delivery, particularly *in vivo*.[198] While *in vitro* methods including ribonucleoprotein transfection and lentiviral transduction can achieve relatively rapid, efficient, and error-free delivery, *in vivo* strategies remain limited by low delivery efficiencies.

Several different methods will deliver Cas9/sgRNA nucleases into cells *in vitro*. Genes for Cas9 and sgRNA can be delivered by plasmid transfection or viral transduction. Transfection of plasmids expressing Cas9 and the sgRNA was the first successful CRISPR delivery strategy, and remains one of the simplest;[168] but the need for cellular Cas9 and sgRNA expression leads to longer experiments, and extended expression increases the rate of off-target DNA cleavage.[199] Viral transduction is useful for screening a library of sgRNA because the viral vector integrates into the genome and provides an sgRNA barcode of cells with the phenotype of interest.[181] However, making viral particles is time-intensive, and extended expression causes similar off-target effects to plasmid transfection. Co-delivery of Cas9 mRNA and sgRNA, either via microinjection or transfection, achieves high cleavage efficiencies with low off-target effects; however, to achieve high cleavage efficiency requires chemical modification of the sgRNA.[200, 201, 202] Direct delivery of Cas9/sgRNA ribonucleoprotein (RNP) complexes is a promising strategy. RNP complexes have been delivered via electroporation and transfection using cationic lipids.[203, 204] They act rapidly and transiently in the cell, leading to high efficiency gene editing with few off-target effects.

Several approaches have been pursued for *in vivo* delivery of CRISPR gene editing components. Hydrodynamic injection, in which a large volume of solution containing Cas9+sgRNA genes and donor template DNA is rapidly injected into the bloodstream so that hydrodynamic forces permeabilize the membranes of endothelial cells to DNA, was used to treat murine liver hepatocytes with hereditary tyrosinemia.[196] Paired adeno-associated viral (AAV) vectors, one containing the Cas9 gene and the other the sgRNA gene and donor template, have been used to deliver CRISPR machinery to correct a urea cycle disorder in liver hepatocytes of baby mice.[205] Cas9 mRNA has been delivered in lipid nanoparticles, in conjunction with an AAV vector containing the sgRNA gene and donor template DNA, to treat liver hepatocytes with tyrosinemia in adult mice.[206] Finally, cationic lipid transfection reagents have been mixed with Cas9/sgRNA RNP complexes and spread on the inner ear of a murine model, with GFP knockout observed in outer hair cells.[204]

While the progress made on *in vivo* delivery of CRISPR/Cas9 is promising, there are clear drawbacks and challenges to the strategies that have been pursued so far. Chief among them is delivery efficiency: only a small fraction of cells actually undergo gene editing, even in the target organ. Hydrodynamic injection, for instance, only corrected one in every 250 hepatocytes.[196] The dual-AAV delivery strategy and the AAV/lipid nanoparticle performed somewhat better, modifying 10% and 6% of hepatocytes respectively.[206, 207] Delivery of Cas9/sRNA complexes with cationic lipid edited only 20% of outer hair cells, right on the surface of the inner ear.[205] For comparison, editing efficiencies of mouse zygotes *in vitro* can approach 90%.[207] Other challenges are specific to the different techniques. Hydrodynamic injection is hazardous, as the excess of liquid impairs cardiac function.[208] AAV vectors are efficiently taken up by cells, but they also generate a humoral, acquired immune response that makes repeated delivery difficult.[209] Nonviral delivery methods are promising because they are non-immunogenic and thus repeated treatments are more feasible; however, high doses of cationic lipids are cytotoxic and provoke inflammatory responses.[210]

An alternative CRISPR/Cas9 delivery strategy, which overcomes some of these challenges, is needed. Based on previous research, a mechanism for delivering Cas9/sgRNA RNP complexes could enable rapid and efficient editing without off-target effects. A nontoxic, non-immunogenic, non-viral nanoparticle would enable repeated gene editing treatments without provoking an acquired immune response. To maximize efficacy, these particles should be stable in serum and rapidly taken up by mammalian cells, and should have a track record of delivering macromolecules into the cytosol.

Spherical nucleic acids (SNAs), nanoparticles modified with a shell of radially oriented oligonucleotides, have many of the desired properties of an improved CRISPR delivery vehicle. (**ref**) The highly oriented oligonucleotide shell gives these nanoparticles (which can have cores made of gold, liposomes, proteins, or other materials) rapid endocytosis into mammalian cells, low immunogenicity, and resistance to nucleases.[211] SNAs have been used as a delivery platform for DNA and RNA cellular diagnostics and therapeutics, including hybridization-based intracellular biosensors of mRNA (NanoFlares) and gene regulation via transfection-agent-free delivery of siRNA.[77, 82] Both these applications demonstrated that SNAs are able to escape the endosome, because the mRNA targets upon which both NanoFlares and siRNA-SNAs act are in the cytosol.[86]

However, an understanding of the mechanisms and parameters that govern the escape of SNAs from the endosome are missing. This is a particularly important knowledge gap, as increased efficiency of endosomal escape could greatly increase the efficacy of SNA therapeutics. It is possible that delivery of

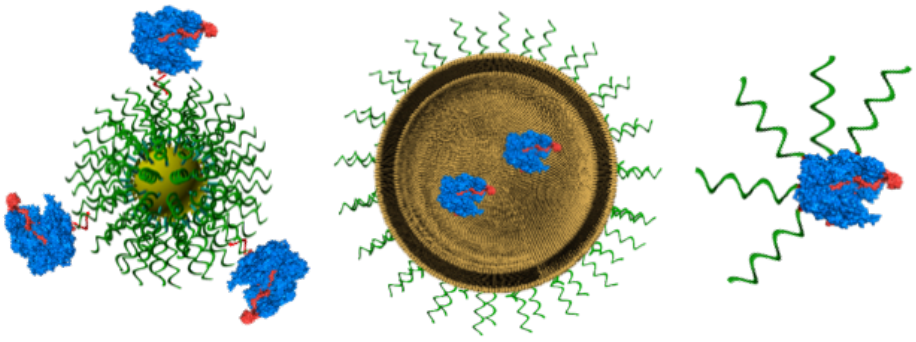


Figure 58. Potential SNA-mediated CRISPR delivery methods. Cas9/sgRNA RNPs could be hybridized to the surface of SNAs (left), encapsulated in a liposomal SNA (center), or chemically modified with oligonucleotides to form a protein-core SNA.

Cas9/sgRNA into cells could serve as a sensitive measure of endosomal escape, since the rate of cytosolic delivery could be correlated with the rate of genomic editing in the cell population. Chapter 4 explores the efficacy of spherical nucleic acids as a strategy to efficiently deliver SpCas9/sgRNA CRISPR RNPs into cells, and of RNP delivery as a means to study the cell uptake and endosomal escape of SNAs.

4.2 Potential SNA-mediated CRISPR delivery methods

There are three mechanisms by which protein might be combined with SNAs and delivered into the cytosol: attachment to an SNA's surface, direct chemical modification with DNA to form a protein-SNA, or encapsulation in the core of a liposomal SNA **(Figure 58)**. Protein directly modified with DNA to form a protein-SNA (proSNA) has been verified to deliver active enzymes into cells.[95] However, it is not known whether those enzymes escaped into the cytosol. Moreover, the enzyme previously delivered, ß-galactosidase, acts on a small molecule substrate. Cas9 undergoes extensive conformational changes to bind both its sgRNA and a double stranded DNA molecule, and may not tolerate being extensively functionalized with DNA while maintaining endonuclease activity. SNAs have been constructed in which a protein or peptide is modified with a single DNA strand that then hybridizes to the SNA's surface; these constructs have also been shown to enter cells.[90, 94] However, this approach exposes the protein to the serum, either *in vivo* or in the media for *in vitro* experiments, possibly increasing the particle's immunogenicity and reducing its stability. Attaching too many proteins to an SNA's surface may also reduce cellular uptake. Liposomal encapsulation of protein does not require extensive modification of the encapsulated protein,[212] and protects the protein from the external environment, potentially enabling full use of SNAs' low immunogenicity. Moreover, an encapsulation strategy enables diverse stability and escape properties to be probed by varying the LSNA's liposome composition and the DNA attachment chemistry.

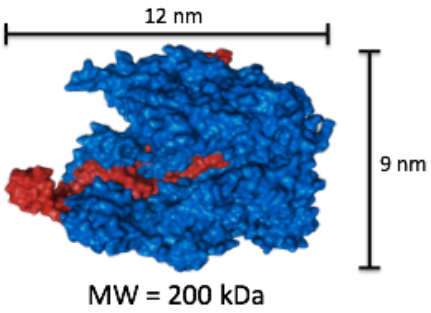


Figure 57. Cas9/sgRNA ribo-nucleoprotein complex. Cas9 (blue) binds sgRNA (red).

4.3 Exploring Cas9/sgRNA attachment to the surface of SNAs

Previous work has attached protein (specifically, immunoglobulins) to the surface of an SNA by chemically modifying the protein with a DNA oligonucleotide strand, and hybridizing the DNA oligonucleotide to the SNA. We hypothesized that SpCas9 would not need to be directly modified, because it binds to an oligonucleotide (its sgRNA) with picomolar affinity, and because additional RNA sequence has been added to the 3' end of the sgRNA without affecting Cas9/sgRNA function. Therefore an attachment strategy was pursued in which a 12 base 3' linker sequence was added to the sgRNA, and this linker sequence was then hybridized to the DNA strands on the SNA.

If SNA surface-attached Cas9/sgRNA complexes do enter cells, the Cas9/sgRNA complex may need a way to dissociate from the nanoparticle in order to enter the nucleus and cleave DNA. Two sets of sgRNA-SNA linker systems were therefore designed: one in which a ubiquitously expressed mRNA (gamma-actin) could hybridize to the SNA and displace the sgRNA, and one in which the linker sequence was scrambled so no mRNA would be likely to hybridize and displace the sgRNA. Three SNA and three sgRNA sequences were designed to pursue this strategy (**Table B1**). The first SNA sequence, called the Hyb_SNA_F-Actin (or Act SNA), was terminated by 12 bases which are complementary to gamma-actin mRNA, and are based on a Nanoflare positive control from Prigodich et al.[78] The second SNA sequence, called Hyb_SNA_Scrambled_Actin (or Scr SNA), was terminated by a scrambled version of the last 12 bases in the Act SNA sequence (scrambling performed by the online siRNA Wizard widget). The three sgRNAs were named Hyb_sgRNA_Act_eGFP (or GFP-Act), Hyb_sgRNA_Scr_eGFP (or GFP-Scr), and Hyb_sgRNA_Scr_AAVS1_1 (or NT-Scr). The GFP-Act sgRNA targets GFP and has a 12 base 3' linker identical to a region of gamma-actin mRNA. The GFP-Scr sgRNA targets GFP and has a linker which is complementary to the scrambled sequence on the Scr SNA. Finally, the NT-Scr

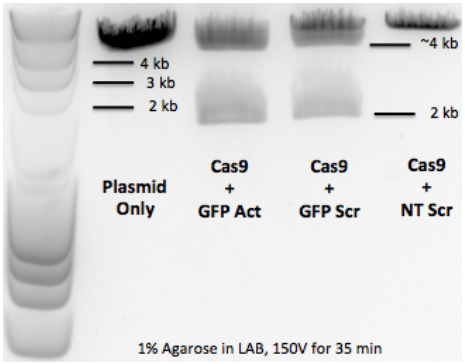


Figure 59. *In vitro* DNA cleavage with Cas9 and designed sgRNAs.

sgRNA targets the human AAVS1 gene and has the same scrambled linker as GFP-Scr. Synthesis of the SNAs proceeded as described in the literature. DNA loading was measured using a Quant-iT Oligreen assay, showing that Scr SNA had ~200 strands per particle, and Act SNA had ~140 strands per particle. Particle size was measured using a Malvern Dynamic Light Scattering (DLS) instrument, indicating a diameter of 18 nm for Act SNA, and 19 nm for Scr SNA, compared to 11 nm for unmodified gold nanoparticle.

The sgRNAs were synthesized using the *in vitro* transcription protocol from Rouge et al.[213] After *in vitro* transcription, sgRNA synthesis was verified using PAGE, and concentration measured via UV-Vis. Each sgRNA was mixed with SpCas9 and tested for activity with an *in vitro* cleavage assay (**Figure 59**). Cas9/sgRNA complexes with the GFP-Act and GFP-Scr sgRNAs cleaved the EGFP gene, while the NT-

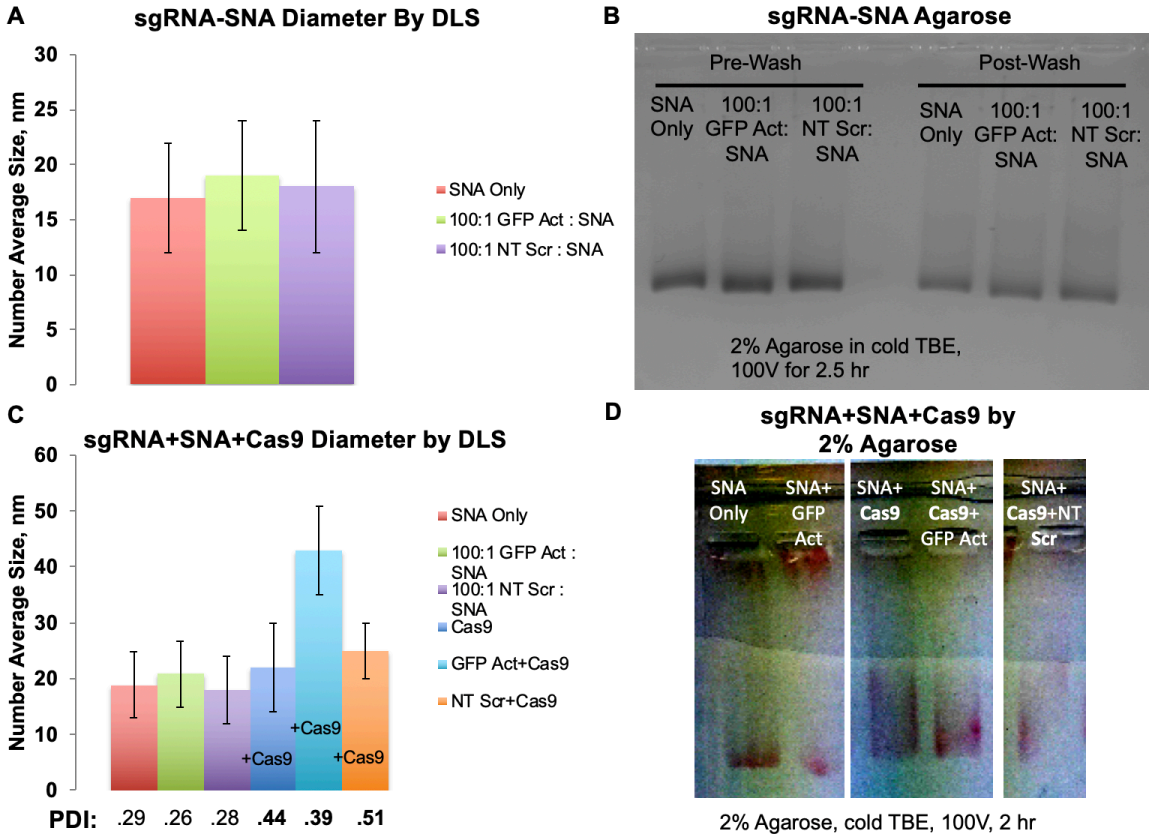


Figure 60. Characterizing Cas9/sgRNA attachment to SNAs. (**A**) DLS number average of SNA diameter before and after adding sgRNA. (**B**) Agarose gel of SNAs after sgRNA hybridization. (**C**) DLS number average and polydispersity index (PDI) of SNA diameter after adding sgRNA and Cas9. (**D**) Agarose gel of SNAs after incubating with sgRNA and Cas9 for 12 hours.

Scr sgRNA did not. This confirmed that the GFP-targeting sgRNAs were still active after the 3' linkers had been added.

We next tested whether the Cas9/sgRNA ribonucleoprotein would bind to the SNAs in a hybridization-dependent manner (**Figure 60**). These experiments were performed first on Act SNA. An incubation of 10 nM SNA with no sgRNA, with 1 μM of either GFP-Act or NT-Scr sgRNA for 12 hours in PBS at 37ºC, then added either nothing or 100 nM SpCas9, and incubated for 2 hours. Particle size was measured by dynamic light scattering (DLS) and by running the particles in a 2% Agarose gel at 100V for 2 hours. While it was expected that GFP-Act to bind the SNA and NT-Scr not to, no increase in particle size and no gel shift on agarose was observed when either sgRNA was added alone. A substantial increase in the polydispersity of the particles (indicating aggregation) was observed when SpCas9 was added to the SNAs, whether or not sgRNA was present. Mixing SpCas9 with the SNAs led to smearing of SNA bands on the agarose gel regardless of the presence of sgRNA, which indicates nonspecific binding of SpCas9 to the SNA. The nanoparticle solution also visibly precipitated in the hours after SpCas9 was added.

SpCas9 most likely binds SNAs nonspecifically because it is highly positively charged. In order to dynamically interact with sgRNA and a DNA duplex, SpCas9 has a clamshell shape, and its interior channel is lined with lysine and arginine residues. The surface of a spherical nucleic acid meanwhile is covered with negative charge from the oligonucleotides' phosphodiester backbones. SpCas9's clamshell structure may enable it to bind very tightly to such a negative surface, shield much of the SNA's charge, and reduce the particle's colloidal stability, leading to the observed precipitation. Whatever the explanation, SpCas9's nonspecific binding of SNAs makes the characterization of any Cas9/sgRNA particles which could actually hybridize to the SNA difficult. Combined with SpCas9's tendency to precipitate the nanoparticles and the potential immunological drawbacks of exposing unmodified SpCas9 to serum for *in vivo* delivery, this led us to pursue other approaches to SNA-mediated Cas9/sgRNA delivery.

4.4 Exploring direct modification of Cas9 to form CRISPR proSNAs

We next sought to determine the ability of Cas9 to function as the core of a protein SNA, while maintaining its enzymatic activity. The most well-validated strategy to attach DNA directly onto protein requires irreversible modification of the primary amines (lysine residues and N-terminus) with an N-hydroxy succinimide ester-oligoethylene glycol-azide (NHS-OEG-azide) moiety that enables alkyne-modified DNA to covalently attach via copper-free click chemistry.[95] We also modified some lysine

Figure 61. Cas9 activity test after Alexa-647 Modification. Numerical ratios represent a dilution series of Cas9, to see activity at different Cas9 concentrations. Negative control band uses NT Scr sgRNA.

residues with an NHS ester-fluorophore conjugate, to make detection and quantification of SpCas9 easier in later experiments. SpCas9, as a DNA and RNA binding protein, has many lysine residues which are important for stabilizing its interaction with highly anionic nucleic acids.[166] Modification with NHS-OEG-azide converts lysines from cationic to uncharged groups, and may disrupt these interactions and SpCas9's enzymatic function, or even destabilize its folding.

To get a preliminary sense of how much lysine modification Cas9 can tolerate, we performed a fluorophore modification experiment. For fluorophore modification, Alexa 647 was chosen because it is highly soluble, and because very few other molecules measurably absorb at 647 nm, so it is relatively easy to detect and quantify in a mixture. 1 mL of 5 μM SpCas9 in PBS was mixed with a 50 μM final concentration of NHS ester-conjugated Alexa 647 and the reaction was incubated overnight at 4ºC. Excess fluorophore was washed away by buffer exchanging 5 times in an Amicon 50k filter column. The absorbances at 280 nm and 647 nm and divided by extinction coefficients produced the final concentrations of both SpCas9 and Alexa, and therefore the number of Alexa modifications per protein. Modification

reactions generally conjugated ~2 Alexa fluorophores per protein, with a low of 0.7 and a high of 5 (data not shown).

After modification, the *in vitro* Cas9 cleavage assay was performed as described above on a dilution series of Cas9, to try to detect any loss of DNA cleavage activity due to modification **(Figure 61)**. We ran 50 μL linearized pcDNA3-EGFP digest tests with 1 picomole, 0.1 picomole, and 0.01 picomole unmodified Cas9+GFP sgRNA, or the same amount of Alexa-Cas9+GFP sgRNA. The gel results show either no decrease or a slight decrease in cleavage activity, depending on the experiment. This indicates that SpCas9 largely tolerates a small number of lysine modifications, but it may not tolerate a larger number.

To test whether SpCas9 can tolerate many lysine modifications, one nanomole of ~1 μM SpCas9 was incubated with a large excess (~480 nanomoles) of NHS-OEG-azide, as described in Brodin et al.[95] A subsequent in vitro cleavage assay showed that the heavily modified SpCas9's activity had been completely abolished (**Figure 62**). A second azide modification was performed on pre-mixed Cas9/sgRNA complexes that had incubated together for an hour, in order to determine if SpCas9 binding to the sgRNA in any way protected essential lysine residues from modification. The Cas9/sgRNA complex's activity was also abolished. These experiments show that while SpCas9 tolerates a small number of irreversible lysine modifications that enable active Cas9 to be labeled and tracked with NHS-fluorophores, it cannot remain active when a large number of its lysine residues are modified; making direct Cas9 modification a less promising approach for delivering active CRISPR RNPs into cells' cytosol.

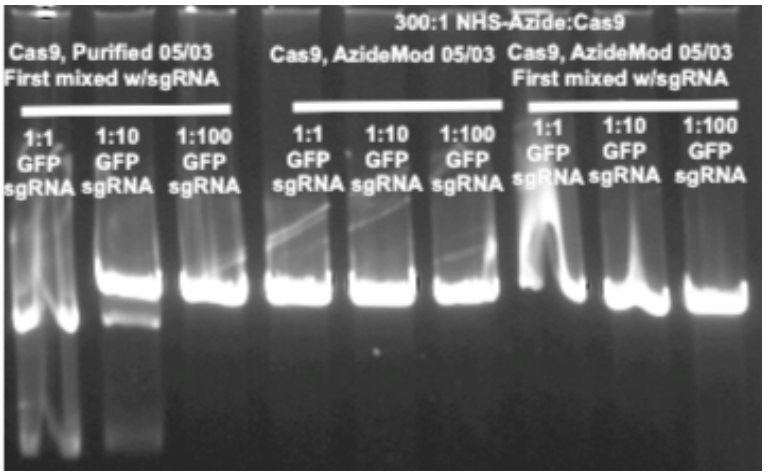


Figure 62. **Cas9 activity test after NHS-azide modification**. Three bands to the left are unmodified Cas9; three middle bands use Cas9 mixed with NHS-azide; three bands to the right use pre-assembled Cas9/sgRNA RNPs mixed with NHS-azide

4.5 Cas9/sgRNA encapsulation in liposomal CRISPR SNAs

The third approach we explored SNA-mediated CRISPR delivery was RNP encapsulation in liposomes. We planned to synthesize liposomal CRISPR SNAs with a range of phospholipid compositions, in order to explore different potential mechanisms of endosomal escape. For liposomal SNAs in particular, there are two primary mechanisms by which macromolecules could be delivered into the cytosol: lysis of the endosomal membrane, or fusion with the endosomal membrane. Endosomal membrane lysis is hypothesized to be facilitated by the "proton sponge effect," in which polycations act as buffers in the late endosome and lysosome, increasing the flow of protons and counterbalancing chloride ions into the vesicles, causing lysis due to osmotic pressure.[214, 215] Liposomes containing dioleoyldiaminopropane (DODAP), a cationic lipid with a pKa of 5.8, could facilitate endosomal lysis by this mechanism. Fusion of liposomes with the endosomal membrane is primarily facilitated by mixing a bilayer-forming lipid like

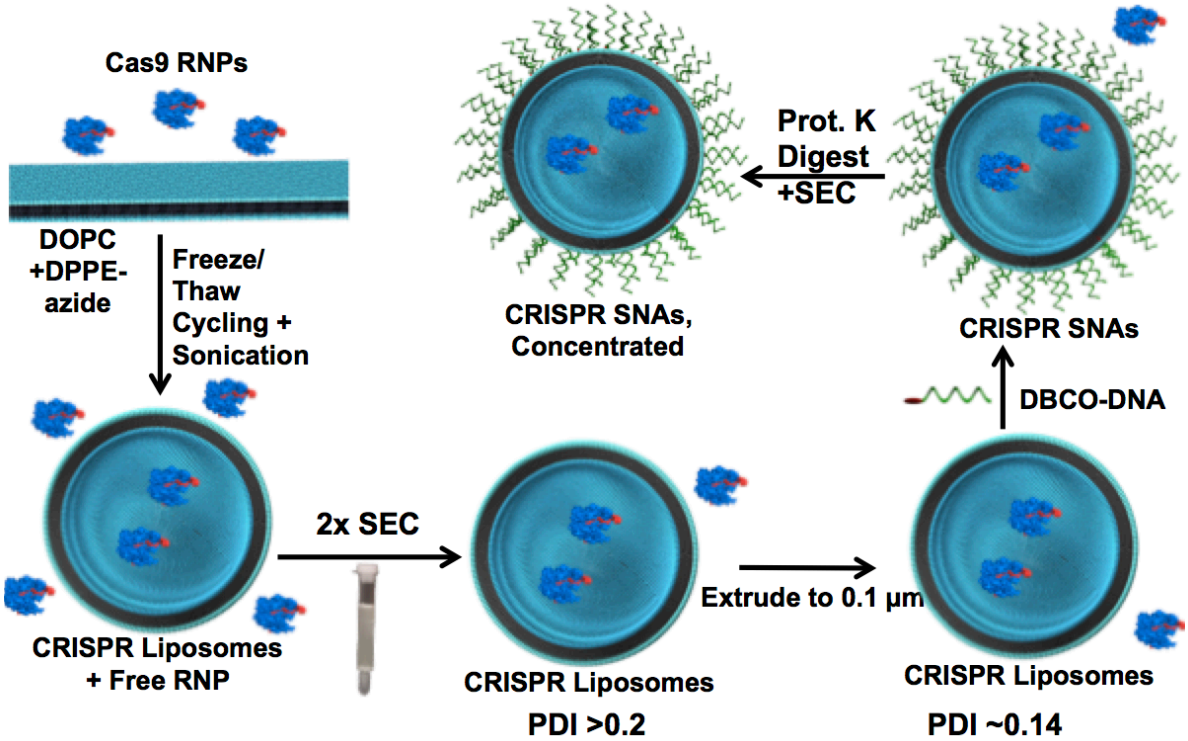


Figure 63. Synthesis of Liposomal CRISPR-SNAs. Concentrated Cas9 RNPs are encapsulated in liposomes, most unencapsulated RNPs are removed via SEC, liposomes were extruded to reduce polydispersity, DBCO-DNA is added to functionalize liposomes with DNA, liposomes are incubated with proteinase K to digest remaining unencapsulated Cas9, and finally digested Cas9 is removed via SEC.

DOPC with lipids that do not stably form bilayers or liposomes on their own, such as dioleoylphosphatidylethanolamine (DOPE).[216] However, the two membrane bilayers must touch in order for fusion to occur. However, for the initial experiments, we synthesized liposomes primarily containing the non-fusogenic phospholipid dioleoylphosphatidylcholine (DOPC).

To construct and test liposomal CRISPR SNAs, we designed new sgRNA and SNA sequences (**Table A2**). We pursued a covalent oligonucleotide attachment strategy to the liposomes, by doping the liposomes with 1% dipalmitoylphosphaditylcholine-azide (DPPE-azide) and then incubating with a dibenzocyclooctyne (DBCO)- and Cy3-modified DNA oligonucleotide (DBCO-Cy3-DNA). To increase the scale of sgRNA synthesis and increase sgRNA chemical stability, we synthesized split crRNA and tracrRNA sequences with phosphorothioate and 2'O-methyl modifications that make the RNA resistant to

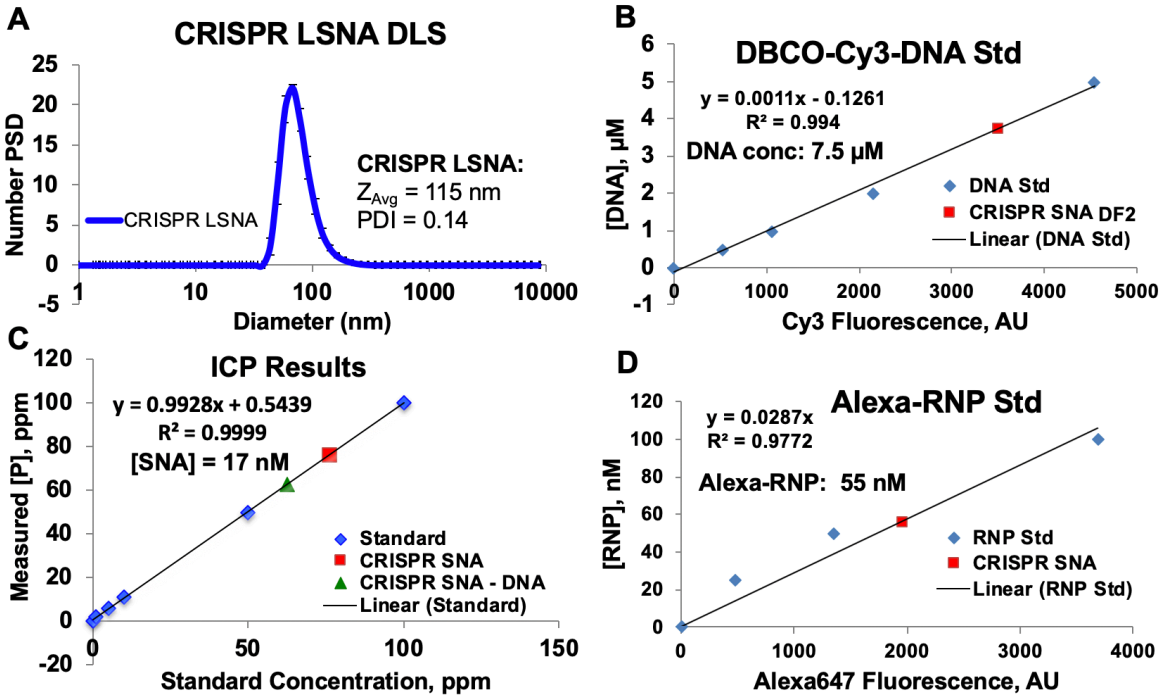


Figure 64. Quantification of DNA and RNP loading in liposomal CRISPR SNAs. (**A**) DLS of CRISPR SNAs after DNA functionalization and cleaning. (**B**) Standard curve of Cy3-DNA fluorescence, with SNA sample (diluted by half). (**C**) ICP-OES quantification of phosphorus (and therefore phospholipid) concentration in CRISPR SNA sample, including standard curve (blue), SNA sample (red), and SNA sample after correcting for the concentration of DNA obtained in B. SNA concentration is calculated using equation 1. (**D**) Standard curve of Alexa647-RNP fluorescence, with SNA sample (blue) plotted with a linear fit.

nucleases without abolishing Cas9 activity.[202] In addition to a GFP-targeting crRNA, we synthesized crRNAs with guide sequences targeting the human EMX1 and FANCF genes, which have previously been validated as model target loci to quantify CRISPR gene editing efficiency.[205] To make RNPs, crRNA and tracrRNA were hybridized together before mixing with the Cas9. We developed a synthesis strategy for making and cleaning liposomal CRISPR SNAs (**Figure 63**), which involved rehydrating a lyophilized lipid film with concentrated RNP solution, running this rehydrated solution through several freeze/thaw cycles to generate single unilamellar vesicles (SUVs), running this solution through size exclusion columns to remove most un-encapsulated RNPs, extruding the liposomes through a 0.1 μm filter, functionalizing with DBCO-modified DNA, and finally incubating with proteinase K and size-excluding again to remove any remaining unencapsulated RNPs.

We developed a set of experiments to quantify the concentration of liposomes, surface-functionalized oligonucleotides, and RNPs on the synthesized liposomal CRISPR SNAs. (**Figure 64**). We measured liposome concentration by measuring phospholipid concentration using inductively coupled plasma optical emission spectrometry (ICP-OES), correcting for phosphorus from the functionalized oligonucleotides, and then dividing the phospholipid concentration by the approximate number of phospholipids per liposome, using the equation in **Figure 65**. The concentration of oligonucleotides was measured in a plate reader by treating SNA samples with 0.1% Tween 20 detergent (to disrupt the liposomes and disperse the oligonucleotides), and comparing Cy3 fluorescence in SNA samples to a standard curve generated from

$$C_{liposomes} = C_{DOPC}\,\frac{4\pi\left(\frac{d}{2}\right)^2 + 4\pi\left(\frac{d}{2} - h\right)^2}{\alpha}$$

$$C_{DOPC} = Conc.\,of\ DOPC\ (\mu M)$$

$$= ICP_{OES}\,(ppm) * \frac{32.2\mu M\ DOPC}{ppm}$$

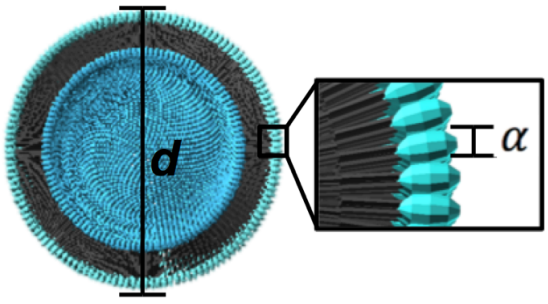


Figure 65. Equation for calculating liposome concentration. D is the diameter (Z average) of the liposomes (or Z average of the SNAs, minus 5 nm for the DNA shell). Alpha (α) is the footprint of the lipid head group, which for DOPC = 0.72 nm^2.

free DBCO- and Cy3-labeled oligonucleotides. The concentration of liposomes was determined with ICP-OES as above, with phosphorus concentration corrected based on the concentration of oligonucleotides and the number of phosphorus atoms per oligonucleotide. The concentration of RNPS was determined by measuring Alexa 647 fluorescence from the liposome samples, and then plotting it on the linear regression of the Alexa-RNP standard curve in a plate reader. In a representative synthesis, we generated 115 nm CRISPR SNAs with ~450 DNA strands per particle, and encapsulated ~3 RNPs per liposome.

We tested whether CRISPR RNPs could remain active through the liposomal CRISPR SNA synthesis (**Figure 66**). We incubated 200 nanograms linearized plasmids with the 1 pmol and 0.1 pmol Alexa RNP immediately after making them, after freeze/thaw cycling, after size exclusion, and after extrusion. The RNPs maintain activity at all stages of CRISPR SNA synthesis.

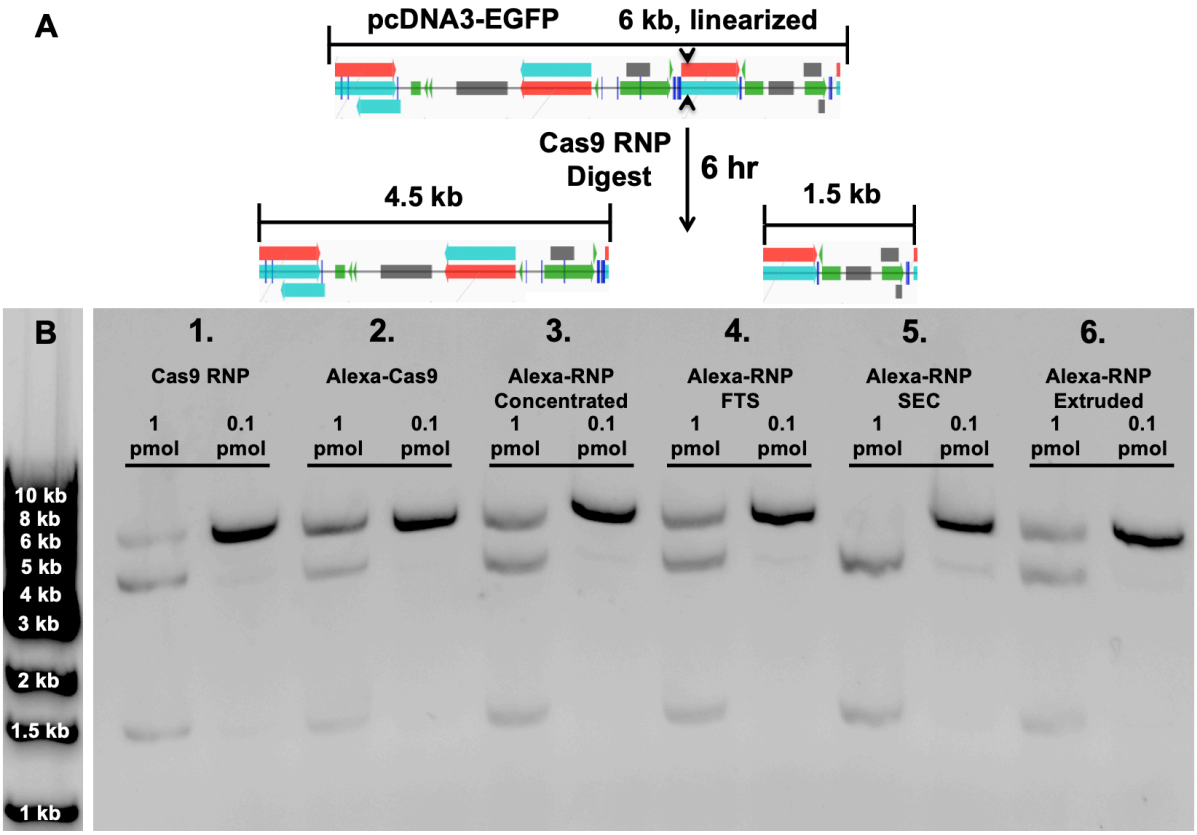


Figure 66. RNPs remain active throughout SNA synthesis procedure. (**A**) Schematic of the *in vitro* Cas9 activity test. (**B**) Activity tests of fresh Cas9 RNPs (**B1**), Cas9 RNPs that have been modified with Alexa dye (**B2**), then concentrated with Amicon 10K filters (**B3**), then subjected to 7 cycles of freeze/thaw/sonication (**B4**), then run through Sepharose 6b SEC columns (**B5**), then extruded 3X through 0.2 μM and 0.1 μM PES membranes (**B6**).

We next sought to determine if RNPs were actually encapsulated in the clean liposomal CRISPR SNAs (**Figure 67**). Clean CRISPR SNAs, and empty SNAs mixed with RNPs, were incubated with proteinase K. Then, both samples were run through a size exclusion column and imaged for DBCO-Cy3-DNA and Alexa 647-modified RNP fluorescence. The unencapsulated RNPs were clearly degraded, as their Alexa-647 fluorescence elutes separately from the Cy3 signal of the liposomal SNAs. By contrast, the fluorescence of

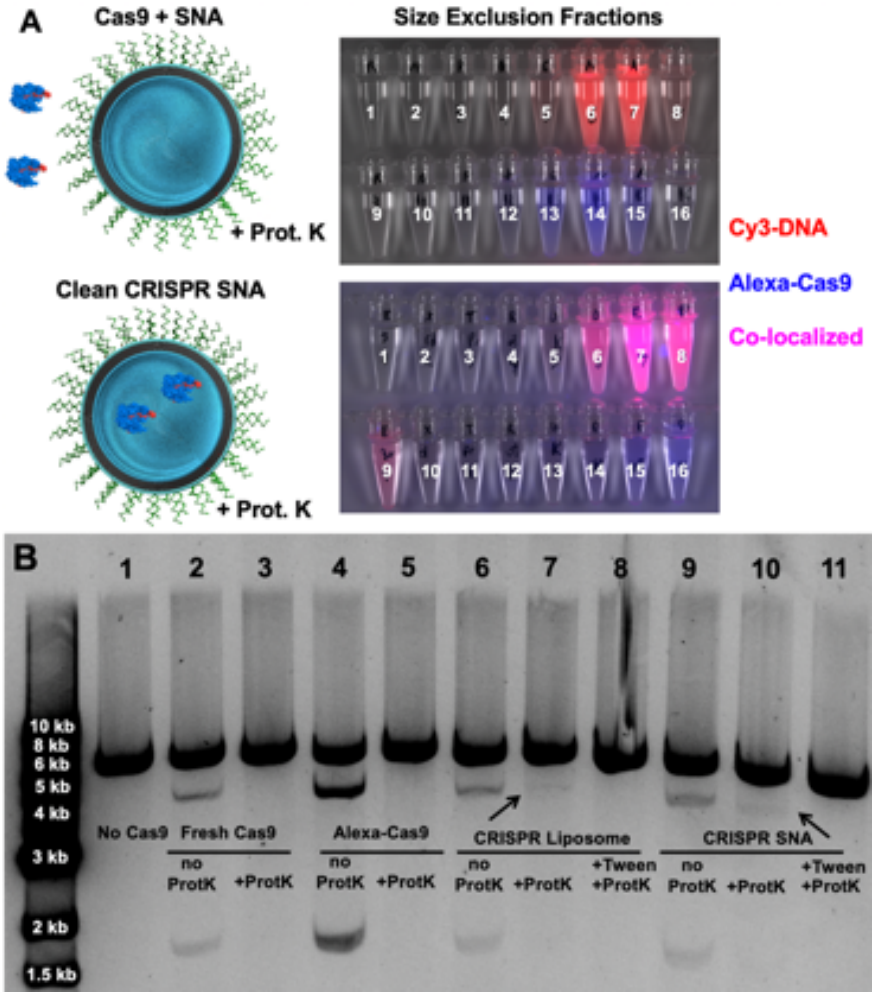


Figure 67. CRISPR-SNAs protect active RNPs from protease, indicating encapsulation. (**A**) Size exclusion fractions collected from a Superdex 200 column after incubating proteinase K with a mixture of empty SNAs and Alexa-RNPs (top) or CRISPR SNAs with encapsulated Alexa-RNPs (bottom). Cy3 (DNA) fluorescence is shown in red, Alexa647 (Cas9) fluorescence in blue, and co-localization of Cy3 and Cas9 fluorescence in pink. (**B**) *In vitro* Cas9 activity tests were run with no Cas9 (**1**); fresh Cas9 without proteinase K (**2**) and with proteinase K (**3**); Alexa-modified Cas9 without proteinase K (**4**) and with proteinase K (**5**); CRISPR liposomes without proteinase K (**6**), with proteinase K (**7**); and with proteinase K added after disrupting liposomes with Tween 20 (**8**); and finally, CRISPR SNAs without proteinase K (**9**), with proteinase K (**10**), and with proteinase K added after disrupting liposomes with Tween 20 (**11**).

the RNPs in the CRISPR SNAs co-elute with the Cy3 signal of the liposomal SNAs, suggesting that the RNPs remain protected and encapsulated in the liposomes after the protease digestion. Further, when the CRISPR SNA samples are mixed with Tween 20 detergent to disrupt the liposomes, Cas9 cleavage activity is measurable in the *in vitro* cleavage assay after proteinase K digestion of the CRISPR SNAs, while the cleavage activity of Cas9/sgRNA RNPs mixed with empty SNAs was abolished after proteinase K digestion.

4.6 Liposomal CRISPR SNAs in cells

We next tested whether CRISPR SNAs could deliver Cas9/sgRNA RNPs into mammalian cells (**Figure 68**). We incubated C166-GFP cells with CRISPR SNAs, empty SNAs, RNPs encapsulated in bare liposomes, and RNPs complexed with RNAiMAX transfection reagent, for 16 hours in Opti-MEM reduced

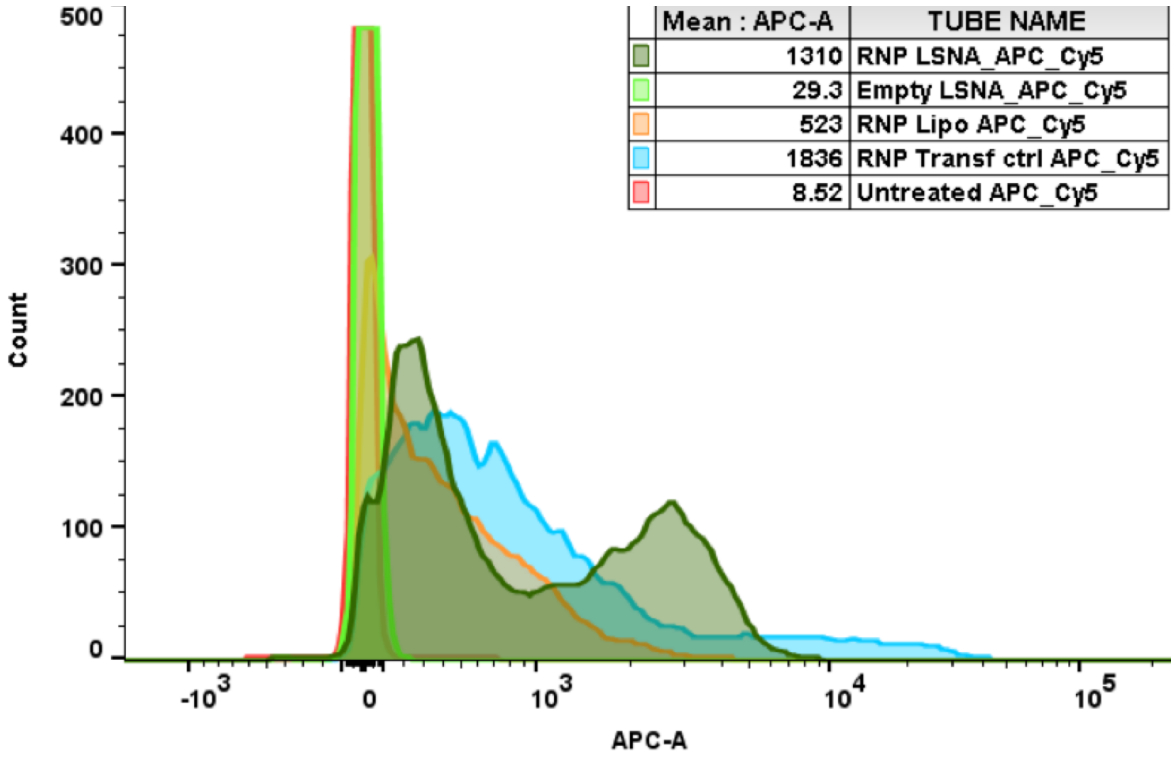


Figure 68. CRISPR-SNAs are actively taken up into mammalian cells. After incubating 5 picomole-equivalents of Alexa RNP of each sample with C166-GFP cells for 16 hours, Alexa 647 fluorescence measured on the allophycocyanin (APC) excitation and emission filter. Histogram of Alexa-RNP fluorescence for untreated cells (red, overlaps with Empty LSNA), empty Cy3-modified LSNA (bright green), RNPs encapsulated in liposomes (orange), Alexa-RNPs transfected with RNAiMax, and finally CRISPR SNAs (dark green).

serum media. Uptake of RNPs labeled with Alexa Fluor 647 was then measured via flow cytometry. Cells treated with CRISPR-SNAs had higher median fluorescence and a higher proportion of highly fluorescent (fluorescence >1000 AU) cells than those treated with RNP/RNAiMAX mixtures or RNPs encapsulated in bare liposomes, while untreated cells showed almost no fluorescence. This data indicates that gene-editing enzymes encapsulated in liposomal SNAs are actively taken up into mammalian cells.

We next tested whether liposomal SNA-mediated delivery of Cas9/sgRNA could induce gene editing in mammalian cells (**Figure 69**). CRISPR SNAs with liposomes containing a range of fusogenic DOPE phospholipid (5, 10, and 15%) were synthesized, and 140 nM Cas9/sgRNA in CRISPR SNAs was incubated with HeLa cells in OptiMEM. As a positive control, Cas9/sgRNA complexes were also mixed with Lipofectamine 2000 transfection reagent, and incubated with cells under the same conditions. After 48 hours, we extracted DNA from the cells, amplified the EMX1 or the FANCF locus by PCR (**Table A3**),

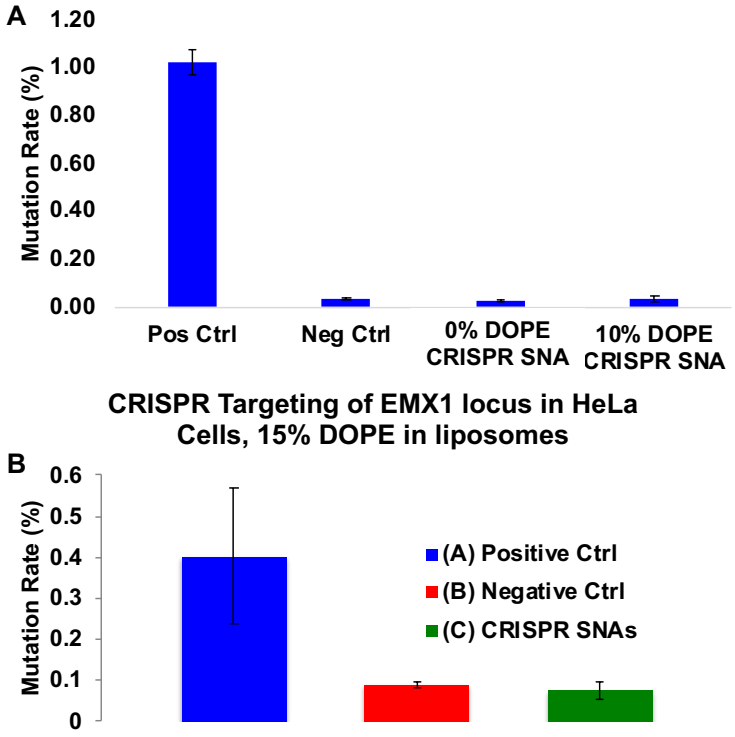


Figure 69. Efficiency of CRISPR-induced mutation at the EMX1 locus in HeLa cells. (**A**) Experiment testing CRISPR SNAs with 0% DOPE and 10 % DOPE liposomes, compared to a positive control of CRISPR RNPs transfected with Lipofectamine 2000. (**B**) Experiment testing the editing activity of CRISPR SNAs with 15% DOPE liposomes.

and sequenced on an Illumina MiSeq. A Matlab script from Kevin Zhao in David Liu's lab was used to calculate the mutation rate from the percent of full FASTQ reads from the Illumina run which contain an insertion or deletion mutation within 5 bases of the predicted Cas9 cleavage site (**Code D1**). Although gene editing was detectable in the transfected positive controls across multiple experiments, gene editing with the CRISPR SNA constructs was not, even when doped with 10% and 15% fusogenic DOPE. So far, SNAs have not been able to successfully deliver gene editing enzymes into mammalian cells.

4.7 Materials and Methods

4.7.1 Materials. Unless otherwise noted, all reagents were purchased from commercial sources and used as received. For oligonucleotide, crRNA and tracrRNA synthesis, all phosphoramidites and reagents were purchased from Glen Research, Co. (Sterling, VA, USA). All lipids were purchased from Avanti Polar Lipids (Alabaster, AL, USA) either in dry powder form or chloroform and used without further purification. EnGen® Cas9 NLS (Cas9) and proteinase K were purchased from New England Biolabs (Ipswich, MA, USA). Alexa Fluor 647 NHS ester dye (Alexa 647) was purchased from Lumiprobe Corp. (Cockneysville, MD, USA). Plasmids were purchased from AddGene (Cambridge, MA, USA. GelRed dye was purchased from Biotium Inc. (Fremont, CA, USA). All other reagents were purchased from Sigma-Aldrich (St. Louis, MO, USA). C166-GFP cells were purchased from ATCC (Manassas, VA, USA), and Opti-MEM was purchased from Life Technologies (Carlsbad, CA).

4.7.2 Cas9 labeling and quantification. In order to track and quantify Cas9, 2 nanomoles of Cas9 were incubated with 10 nanomoles of Alexa 647 NHS Ester, in 1X HBS overnight at 4ºC, generating Alexa-Cas9. To remove unreacted dye, Alexa-Cas9 was run through a NAP5 column equilibrated in 1X HBS, and eluted in 1 mL 1X HBS. 2 nanomoles unmodified Cas9 was exchanged into 1X HBS using a NAP5 column, and combined with the Alexa Cas9. The concentration of Cas9 and Alexa dye were calculated using the absorbance at 280 nm and 650 nm, respectively; and the molar ratio of Alexa dye to Cas9 was calculated. The Alexa-Cas9 is then diluted to 1 μM. A 20 μL aliquot is reserved for activity and concentration assays.

4.7.3 Cas9 ribonucleoprotein synthesis and concentration. 10 nanomoles crRNA and tracrRNA were generated by incubating 10 μM crRNA with 10 μM tracrRNA in 1X HBS at 95ºC for 5 minutes, and allowed to cool to room temperature for 10 minutes. 10 nanomoles of crRNA/tracrRNA complex is then mixed with 4 nanomoles of 1 μM Alexa-Cas9 and allowed to sit at room temperature for 10 minutes, to form the Cas9 ribonucleoprotein (RNP). RNPs were then concentrated in Amicon 10K spin filters. 5 minute stretches, then resuspending, until the retained liquid volume reaches 500 μL or less. Cas9 concentration was again quantified using the absorbance of the Alexa 647 dye. 20 μL were set aside for activity and concentration measurements.

4.7.4 Synthesis and purification of CRISPR SNAs. To synthesize liposomes encapsulating Cas9 RNPs, a dehydrated phospholipid film was generated by lyophilizing a mixture of 3 mg DOPC and 0.15 mg DPPE-Azide in chloroform. The lipid film was then rehydrated with 400 μL of Alexa 647-labeled ribonucleoprotein complexes (Alexa-RNPs) in 1X HBS, at a concentration of 5-8 μM. This solution was then subjected to 7 freeze/thaw cycles using liquid nitrogen and a room-temperature bath sonicator to generate single unilamellar vesicles (SUVs). The SUVs were run through a column packed with Sepharose 6B and equilibrated in 1X HBS to separate them from unencapsulated RNPs. To reduce polydispersity, the SUVs were extruded twice through 200 nm and then 100 nm membrane filters. To remove the remaining unencapsulated RNPs, SUVs were incubated for 1 hour at room temperature with proteinase K (10 U, in 500 μL 1X NEB Buffer 2 + 1X HBS). SUVs were separated from digested RNPs using a column packed with Superdex 200 and equilibrated in 1X HBS. To generate SNAs, the SUVs were then incubated overnight with oligonucleotides functionalized on the 5' end with DBCO and internally with Cy3 (~1 DNA per 20 phospholipids). SNAs were then separated from free oligonucleotides using a column packed with Superdex 200 and equilibrated in 1X HBS.

4.7.5 Quantification of Cas9 and DNA loading. To measure SUV concentrations, inductively coupled plasma optical emission spectrometry (ICP-OES) and a phosphorus standard were used to calculate phospholipid concentration. Liposome diameter was measured via dynamic light scattering (DLS), and the

number of phospholipids per liposome were calculated. SUV concentration was calculated by dividing phospholipid concentration by the number of phospholipids per SUV. The concentration of oligonucleotides was measured in a Bio-Tek H4 plate reader by treating SNA samples with 0.1% Tween 20 detergent (to disrupt the liposomes and disperse the oligonucleotides), and comparing Cy3 fluorescence in SNA samples to a standard curve generated from free DBCO- and Cy3-labeled oligonucleotides. The concentration of liposomes was determined with ICP-OES as above, with phosphorus concentration corrected based on the concentration of oligonucleotides and the number of phosphorus atoms per oligonucleotide. To calculate the concentration of RNPs, a standard curve of Alexa-647 fluorescence was generated from an Alexa-RNP aliquot in the plate reader. The concentration of RNPs was determined by measuring Alexa-647 fluorescence from the liposome samples, and then plotting it on the linear regression of the Alexa-RNP standard curve in a plate reader.

4.7.6 In vitro Cas9 DNA cleavage assay. RNPs targeting the EGFP gene were synthesized and used to make CRISPR SNAs. Purified pcDNA3-EGFP plasmid was linearized by digesting with restriction enzyme SmaI. Active RNPs incubated with the linearized plasmid cleave it into a 2 kb and a 4 kb fragment, which can be seen on a 1% agarose electrophoresis gel run in TBE buffer for 30 minutes. 200 nanograms linearized plasmids with the 1 pmol and 0.1 pmol Alexa RNP immediately after making them, after freeze/thaw cycling, after size exclusion, and after extrusion.

4.7.7 Protease stability studies. To verify that RNPs are encapsulated inside SNAs, clean CRISPR SNAs were incubated with proteinase K in NEB's restriction enzyme buffer 2 for 1 hour at room temperature. As a control, Alexa-RNPs were mixed with empty SNAs and incubated with proteinase K. The incubated samples were then eluted in 200 μL fractions through a Superdex 200 size exclusion column equilibrated in 1x HBS. These fractions were then imaged in a fluorescent gel scanner for Cy3 and Alexa Fluor 647 fluorescence. To verify that the encapsulated RNPs are still active liposomes in CRISPR SNAs were disrupted with 0.1% Tween 20 detergent either before or after incubating them with proteinase K as above.

Then proteinase K was inactivated with 1 mM 1 mM phenylmethylsulfonyl fluoride (PMSF), and the *in vitro* cleavage assay was run as above.

4.7.8 Cell uptake studies. C166-GFP cells were incubated with CRISPR SNAs, empty SNAs, RNPs encapsulated in bare liposomes, and RNPs complexed with RNAiMAX transfection reagent, for 16 hours in Opti-MEM reduced serum media. Uptake of RNPs labeled with Alexa Fluor 647 was then measured via flow cytometry.

4.7.9 Quantification of Gene Editing. CRISPR SNAs were synthesized using several different liposome compositions, including 98% DOPC + 2% DPPE-Azide, 88% DOPC + 10% DOPE + 2% DPPE-Azide, and 83% DOPC + 15% DOPE + 2% DPPE-Azide. Cas9/sgRNA RNPs encapsulated in the SNAs contained an equimolar mixture of crRNAs targeting these loci. HeLa cells were incubated with 140 nM Cas9/sgRNA complexes in CRISPR SNAs (quantified by Alexa 647 fluorescence) for 12 hours at 37ºC in OptiMEM culture media. As a positive control, Cas9/sgRNA complexes were also mixed with Lipofectamine 2000 transfection reagent, and incubated with cells under the same conditions. OptiMEM was removed and replaced with DMEM + 10%FBS, and cells were allowed to grow for 48 hours. Then, genomic DNA from each well was extracted with a Qiagen kit, the sequence for the EMX1 locus was amplified and barcoded by PCR using the Nextera Illumina library preparation kit, and sequenced on an Illumina MiSeq. The mutation rate was calculated from the percent of full reads from the Illumina run which contain an insertion or deletion mutation within 5 bases of the predicted Cas9 cleavage site, using the script **Code D1**.

CHAPTER 5: Outlook, Future Work

5.1 Dual Readout Sandwich Immunoassay

The dual-readout sandwich immunoassay presented in chapter 2 enables both device-free visual and colorimetric analysis of samples, as well as highly sensitive scanometric target detection. Previous work on dual-readout Pt-coated AuNP detection strategies used AuNP local surface plasmon resonance (LSPR) to detect them colorimetrically, and therefore required the careful deposition of only a few layers of Pt atoms onto the AuNPs to avoid disrupting their LSPR.[21] In this approach, the Pt-based colorimetric readout is more sensitive than the Au-based colorimetric detection. By contrast, our assay demonstrates much greater sensitivity through the gold-based readout than the Pt-based one, highlighting the value of the gold amplification step and scanometric detection method. The sub-picomolar scanometric readout's detection sensitivity is comparable to other ultrasensitive nanoparticle-based PA_{83} assays employing europium nanoparticle-based fluorescence and silver nanoparticle-enhanced fluorescence.[103, 104]

One way to improve this assay in the future would be to reduce the time required to get a colorimetric readout. This could be achieved by shortening the platinum deposition step. The platinum metal precursor solution used in this work is based on prior research into the synthesis of colloidally stable Au@Pt nanostructures.[129] However, in our assay, the colloidal stability of the immobilized AuNPs after Pt deposition is unimportant; what matters is that Pt is deposited specifically on the AuNPs, and not on surfaces that lack AuNPs. It may be that a Pt deposition solution with a higher concentration of metal precursor could achieve the same level of AuNP-specific Pt deposition (and therefore colorimetric signal amplification) in a shorter period of time.

Another approach to minimize time-to-readout could be to conjugate antibodies directly to peroxidase-mimicking nanoparticles[217] which are capable of both catalytic H_2O_2 splitting and nucleating gold particle growth from reduced ions. Gold nanoclusters (AuNCs), for instance, can catalyze the splitting of O_2 and subsequent oxidation of TMB when illuminated with visible light.[218] Alternatively, anisotropic platinum nanoparticles (PtNPs) can split H_2O_2 and oxidize TMB without requiring a reduction step, and have been functionalized with immunoglobulins to create model immunoassays.[119] If PtNPs could serve as nuclei for

gold reduction, then mAb-PtNP nanoparticles could provide even more rapid device-free colorimetric antigen detection, while still enabling highly sensitive scanometric detection with the same particles.

The colorimetric detection readout developed in this work has similar sensitivity to colorimetric immunoassays employing the enzyme horseradish peroxidase (e.g. ELISA).[219] Another H_2O_2-decomposing enzyme, catalase, has been combined with gold nanoparticles to achieve ultrasensitive visual detection of protein biomarkers.[220] However, in contrast to most enzymes, Pt remains catalytically active across a wide range of temperature and pH,[119] broadening the range of applications and potentially enhancing the field deployability of similar colorimetric assays. Although less sensitive than the scanometric readout, in principle, the colorimetric readout could be employed in pairwise device-free screens to discover pairs of recognition elements, whether antibodies, aptamers, or hyper-stable designer protein binders;[221] while the scanometric detection method can be used to significantly increase the sensitivity of any discovered sandwich pairs. These paired recognition elements, like the anti-PA_{83} mAb pair discovered using the colorimetric readout, could serve as practical tools for the sensitive, specific, and reproducible detection of anthrax and other disease biomarkers.

5.2 Aptamer NanoFlares

So far for the aptamer NanoFlare project, our main conclusion is that it is it is difficult to know *a priori* how a given aptamer-flare pair will respond to target molecule, because target-dependent aptamer-flare dehybridization depends in complicated ways on the 3-dimensional structure of the aptamer, the aptamer-flare duplex, and the hypothesized aptamer-flare-target intermediate structure. To address this challenge, the flare microarray experiments and the analytical pipeline we've built enable comprehensive exploration of the aptamer-flare design space, and have shown that aptamer structural loops and 5' and 3' termini favor stable aptamer-flare hybridization, as does a calculated ΔG of hybridization stronger than -10 kcal/mol. Even if mathematical models of aptamer NanoFlares cannot rationally predict the 'best' aptamer-flare pair, the microarray experiments and analytical pipeline may make it possible to identify this 'needle in a haystack' anyway.

The next step in this project is to successfully identify target-responsive aptamer-flare pairs from the microarray screens. Toward this end, the microarrays should be redesigned to normalize and correct for several sources of experimental noise that have so far prevented us from identifying target-responsive aptamer-flare pairs from the array data. Specifically, internal hybridization positive controls should be added across the array, that bind a 20 bp fluorophore-modified control oligonucleotide that serves as an internal reference for any changes in absolute fluorescence measured across the array. Another element that has been missing from previous experiments is a detection positive control, a set of aptamer-flare pairs that are known to respond to a target molecule, and can be used to check that the analysis pipeline can actually detect 'hits.' To address this, new microarray designs incorporate all the flares for a known ATP-responsive aptamer that has previously been shown to detectably de-hybridize from a subset of complementary oligonucleotides via an induced fit mechanism in similar microarray experiments.[222] The ATP aptamer positive control will be used to validate and improve the analytical workflow, and then this workflow can be used to screen aptamer-flare pairs responsive to DHEA-S, cortisol, or any other target molecule of interest.

Answering the other questions raised by this project, like what effect attachment to a nanoparticle has on aptamer folding, flare hybridization, and target binding, requires the identification of at least one aptamer-flare pair that consistently detects the target molecule. In addition to continuing the screening experiments to discover such a pair, one place to look for such aptamer-flare 'hits' is the literature. For instance, the Stojanovic group have already reported several aptamer-complementary oligonucleotide pairs that de-hybridize in the presence of different steroid hormones.[160] If at least one of these biosensors could be replicated and validated in a nanoparticle-free, black-hole-quencher-based aptamer fluorescence assay, it could serve as the baseline from which to compare the flare hybridization and target detection behavior of aptamer NanoFlare nanoparticles with different DNA loading densities.

One problem that has stymied such experiments so far is the strange batch-to-batch variability we have observed in DHEA-S detection behavior for our aptamer NanoFlares. One potential source of variability

that hasn't been ruled out yet is the poly-A spacer on the thiolated aptamer sequences. Poly-A DNA oligonucleotides adsorb strongly to the surface of citrate-capped gold nanoparticles; this adsorption of the ostensible spacer sequence in our SNA designs may interfere with aptamer folding, flare hybridization or target molecule binding. In future experiments, the aptamer NanoFlare constructs we have investigated could be synthesized with a poly-T spacer sequence instead,[223] and target detection performance and consistency could be compared to constructs with the poly-A spacer.

A potential limitation of the aptamer NanoFlare architecture we have studied here is that the target molecule must compete with a flare strand that is actively disrupting the aptamer structure in order to bind to the aptamer and stabilize its folded conformation. It is possible that this competition introduces a trade-off between low background fluorescence and target molecule-induced flare dissociation: designing a highly stable aptamer-flare duplex may make it less thermodynamically favorable for the flare to be displaced and replaced with the target molecule. Conversely, designing a highly labile aptamer-flare duplex that is easily disrupted by the target molecule may result in a higher intrinsic aptamer-flare dissociation constant and rate, and therefore a higher concentration of unbound flare in solution even when the target molecule is absent. There is some evidence that this type of target-flare competition reduces the performance of aptamers: in the original ATP aptamer Nanoflare paper, the effective dissociation constant of the aptamer-ATP interaction was 100-fold weaker when the aptamer was hybridized to the flare strand than when there was no flare strand to compete with.[81] This poses a serious challenge to the utility of aptamer NanoFlares as steroid stress biomarkers of DHEA-S or cortisol, since the physiological concentrations of these molecules in saliva (1-10 nM for cortisol;[224, 225] 0.6-70 nM for DHEA-S[226, 227]) are already lower than the dissociation constants of most aptamers for their targets.

One way to address this challenge could be to replace the aptamer NanoFlare design with Mirkin lab's recently reported forced-intercalation (FIT) aptamer design.[228] FIT aptamers don't require a flare strand; instead, one of the aptamer's nucleotides is replaced with fluorescent dye, such as thiazole orange, that selectively fluoresces when sandwiched between and conformationally constrained by the stacked

nucleoside bases of a DNA double helix.[229] If the dye is placed in a location on the aptamer that is not hybridized unless stabilized by binding the target molecule, then it will serve as a fluorescent biosensor of the target. There are many aptamers that alter their structural conformation upon binding their targets; for instance, Mirau lab's NMR data in **Figure 21** shows that the truncated DIS11th_3 aptamer undergoes some sort of conformational change upon binding DHEA-S. It's possible that, in the absence of a competing flare strand, such 'induced fold' FIT aptamers could more sensitively detect their target molecule. SNA architecture could still play an important role in these constructs: for aptamers that fold stably even in the absence of target molecule, perhaps dense packing on the surface of a nanoparticle could destabilize native structure enough to insert a FIT dye with low background fluorescence, while still enabling target molecules to stabilize the aptamer's folded conformation and turn FIT fluorescence back on.

5.3 CRISPR SNAs

The fundamental challenge facing any project aimed at nonviral, cytosolic delivery of proteins *in vivo* remains endosomal escape (for *in vitro* delivery into cell lines, cationic transfection reagents remain useful tools and positive controls). The liposomal CRISPR-SNAs we synthesized had active ribonucleoprotein enzymes inside them, and were endocytosed by cells, but no gene editing was observed. One of the challenges with this result is that because endosomal escape by SNAs is so rare and difficult to quantify, it's not clear by what mechanism the liposomal CRISPR SNAs are failing. It's possible that, despite doping with DOPE, none of the liposomes ever fuse with the endosomal membrane or escape, intact, into the cytosol. It's possible that some liposomes escape into the cytosol intact, but that the Cas9 RNPs never escape from inside them, which they must do to travel to the nucleus and start gene editing. Or it's possible that some liposomes are fusing with the endosomal membrane, and some RNPs are escaping into the cytosol, but the escaped RNP concentration is too low (or the activity of the RNPs is too degraded somehow by the encapsulation process) to detectably mutate genomic DNA. In order to address any of these hypotheses, a sensitive and quantitative method for measuring the endosomal escape of all the nanoparticles' different components is required. We had hypothesized at the beginning of this project that

genomic editing could serve as such a method to sensitively measure endosomal escape; but so far that supposition has not been borne out.

Split GFP-based sensors[230] could be an alternative method to sensitively measure endosomal escape. Fluorescent proteins with one of the beta strands in the beta barrel removed, split GFP variants only fluoresce when complemented by the addition of the missing beta strand into solution. Split GFPs now come in several colors, including a photoactivatable variant that can be used in super-resolution microscopy.[231] Split GFPs have already been used to quantify endosomal escape during the engineering and optimization of cell-penetrating peptides.[232] By expressing the incomplete beta barrel in the cytosol of model cell lines, it's possible that SNAs carrying the complementing beta strand could enter cells and, even when very rare, report endosomal escape events with fluorescence. Just as Shuya Wang et al. investigated the effect of antigen peptide attachment location and mechanism on the efficacy of immune-stimulating SNA cancer vaccines,[92] split GFP beta strands could be encapsulated within, attached via intercalation to, or hybridized onto the oligonucleotide shell of liposomal SNAs with varying phospholipid compositions; and the liposome composition and peptide orientation that causes the most cellular fluorescence could be pursued as a cytosolic peptide/protein delivery vehicle. Once the concept is demonstrated, cytosolic delivery of larger proteins (perhaps even Cas9) could be tracked and quantified by fusing the complementary beta strand to the N or C terminus of the protein to be delivered.

One of the main factors limiting the rate at which different iterations of liposomal CRISPR SNAs could be designed, built and tested was the inefficiency of Cas9 RNP encapsulation inside the liposomes. Because the RNPs were not modified with a phospholipid-bilayer-binding moiety, the enzymes had to be encapsulated by chance. Even at the highest stable RNP concentrations we achieved (roughly 8 μM), only about 2 RNPs would likely be encapsulated by chance inside the volume of a 100 nm liposome. If encapsulation in liposomal SNAs is ever to become a successful and standard method for delivering proteins into cells, the encapsulation efficiency must increase. This could be achieved by modifying the liposome composition and/or fusing a tag to the protein that increases the electrostatic attraction between protein and

phospholipid bilayer; or by chemically conjugating an intercalating moiety to the protein via a labile, bio-reducible bond.

This project also made clear that the standard chemical conjugation method for generating functional protein-core SNAs does not work for CRISPR enzymes. We hypothesize that more generally, irreversible NHS-ester mediated attachment of oligonucleotides to lysine residues is only a viable strategy for creating enzyme SNAs with functional cores if the enzymes in question don't have many structurally important lysines, and don't require large conformational changes that could be blocked by an oligonucleotide shell in order to perform catalysis. Cas9 violates both these criteria, so any future attempts to deliver Cas9 RNPs in proSNA format should investigate how to modify the protein's surface amino acids with traceless, bioreducible linkers. The traceless linker Skakuj et al. have recently applied to SNA-mediated antigen peptide delivery for cancer immunotherapy could merit investigation.[233]

REFERENCES

1. Kreuter, J. Nanoparticles—a historical perspective. *Int. J. Pharm.* **2007**. *331*, 1–10.
2. Yeh, Y.-C.; Creran, B.; Rotello, V. M. Gold nanoparticles: preparation, properties, and applications in bionanotechnology. *Nanoscale*. **2012**. *4*, 1871–80.
3. Jain, P. K.; Lee, K. S.; El-Sayed, I. H.; El-Sayed, M. A. Calculated Absorption and Scattering Properties of Gold Nanoparticles of Different Size, Shape, and Composition: Applications in Biological Imaging and Biomedicine. *J. Phys. Chem. B*. **2006**. *110*, 7238–7248.
4. Bera, D., Qian, L., Tseng, T.-K. & Holloway, P. H. Quantum Dots and Their Multimodal Applications: A Review. *Materials (Basel)*. **2010**. *3*, 2260–2345.
5. Daniel, M.-C.; Astruc, D. Gold Nanoparticles: Assembly, Supramolecular Chemistry, Quantum-Size-Related Properties, and Applications toward Biology, Catalysis, and Nanotechnology. *Chem. Rev*. **2003**. *104*, 293–346.
6. Jazayeri, M. H.; Aghaie, T.; Avan, A.; Vatankhah, A.; Ghaffari, M. R. S. Colorimetric detection based on gold nano particles (GNPs): An easy, fast, inexpensive, low-cost and short time method in detection of analytes (protein, DNA, and ion). *Sens. Bio-Sensing Res*. **2018**. *20*, 1–8.
7. Kim, J.; Biondi, M. J.; Feld, J. J.; Chan, W. C. W. Clinical Validation of Quantum Dot Barcode Diagnostic Technology. *ACS Nano* **2016**. *10*, 4742–4753.
8. Yen, S. K.; Padmanabhan, P.;Selvan, S. T. Multifunctional iron oxide nanoparticles for diagnostics, therapy and macromolecule delivery. *Theranostics* **2013**. *3*, 986–1003.
9. Jazayeri, M. H.; Amani, H.; Pourfatollah, A. A.; Pazoki-Toroudi, H.; Sedighimoghaddam, B. Various methods of gold nanoparticles (GNPs) conjugation to antibodies. *Sens. Bio-Sensing Res*. **2016**. *9*, 17–22.
10. Billingsley, M. M.; Riley, R. S.; Day, E. S. Antibody-nanoparticle conjugates to enhance the sensitivity of ELISA-based detection methods. *PLoS One* **2017**. *12*, e0177592.

11. Li, D.-L. *et al*. Multifunctional superparamagnetic nanoparticles conjugated with fluorescein-labeled designed ankyrin repeat protein as an efficient HER2-targeted probe in breast cancer. *Biomaterials* **2017**. *147*, 86–98.
12. Franco, R.; Pedrosa, P.; Carlos, F. F.; Veigas, B.; Baptista, P. V. Gold Nanoparticles for DNA/RNA-Based Diagnostics. *Handbook of Nanoparticles*. **2015**. 1–25.
13. Xu, H. *et al*. Aptamer-Functionalized Gold Nanoparticles as Probes in a Dry-Reagent Strip Biosensor for Protein Analysis. *Anal*. *Chem*. **2009**. *81*, 669–675.
14. Xia, N.; Chen, Z.; Liu, Y.; Ren, H.; Liu, L. Peptide aptamer-based biosensor for the detection of human chorionic gonadotropin by converting silver nanoparticles-based colorimetric assay into sensitive electrochemical analysis. *Sensors Actuators B Chem*. **2017.** 243, 784–791.
15. Huh, Y.-M. *et al*. In Vivo Magnetic Resonance Detection of Cancer by Using Multifunctional Magnetic Nanocrystals. *J*. *Am*. *Chem*. *Soc*. **2005**. *127*, 12387-12391.
16. Chen, L.; Chen, C.; Li, R.; Li, Y.; Liu, S. CdTe quantum dot functionalized silica nanosphere labels for ultrasensitive detection of biomarker. *Chem*. *Commun*. **2009**. 2670-2672.
17. Hussain, M. M.; Samir, T. M.; Azzazy, H. M. E. Unmodified gold nanoparticles for direct and rapid detection of Mycobacterium tuberculosis complex. *Clin*. *Biochem*. **2013**. *46*, 633–637.
18. Gill, P., Alvandi, A.-H., Abdul-Tehrani, H. & Sadeghizadeh, M. Colorimetric detection of Helicobacter pylori DNA using isothermal helicase-dependent amplification and gold nanoparticle probes. *Diagn*. *Microbiol*. *Infect*. *Dis*. **2008**. *62*, 119–124.
19. Liu, Y. *et al*. Colorimetric detection of influenza A virus using antibody-functionalized gold nanoparticles. *Analyst* **2015**. *140*, 3989–95.
20. Li, Y. *et al*. Horseradish peroxidase-loaded nanospheres attached to hollow gold nanoparticles as signal enhancers in an ultrasensitive immunoassay for alpha-fetoprotein. *Microchim*. *Acta* **2014**. *181*, 679–685.

21. Gao, Z. *et al*. Platinum-Decorated Gold Nanoparticles with Dual Functionalities for Ultrasensitive Colorimetric in Vitro Diagnostics. *Nano Lett*. **2017**. *17*, 5572–5579.

22. Härmä, H.; Soukka, T.; Lövgren, T. Europium Nanoparticles and Time-resolved Fluorescence for Ultrasensitive Detection of Prostate-specific Antigen. *Clin. Chem*. **2001**. *47*, 561–568.

23. Wang, J.; Achilefu, S.; Nantz, M.; Kang, K. A. Gold nanoparticle–fluorophore complex for conditionally fluorescing signal mediator. *Anal. Chim. Acta* **2011**. *695*, 96–104.

24. Choi, D. H. *et al*. A dual gold nanoparticle conjugate-based lateral flow assay (LFA) method for the analysis of troponin I. *Biosens. Bioelectron*. **2010**. *25*, 1999–2002.

25. Rauta, P. R.; Hallur, P. M.; Chaubey, A. Gold nanoparticle-based rapid detection and isolation of cells using ligand-receptor chemistry. *Sci. Rep*. **2018**. *8*, 2893.

26. Harris, N.; Ford, M. J.; Cortie, M. B. Optimization of Plasmonic Heating by Gold Nanospheres and Nanoshells. *J. Phys. Chem. B* **2006**. *110*, 10701–10707.

27. Hirsch, L. R. *et al*. Nanoshell-mediated near-infrared thermal therapy of tumors under magnetic resonance guidance. *Proc. Natl. Acad. Sci. U. S. A*. **2003**. *100*, 13549–54.

28. Vines, J. B.; Yoon, J.-H.; Ryu, N.-E.; Lim, D.-J.; Park, H. Gold Nanoparticles for Photothermal Cancer Therapy. *Front. Chem*. **2019**. *7*, 167.

29. Kruse, A. M., Meenach, S. A., Anderson, K. W. & Hilt, J. Z. Synthesis and characterization of CREKA-conjugated iron oxide nanoparticles for hyperthermia applications. *Acta Biomater*. **2014**. *10*, 2622–2629.

30. Akbarzadeh, A. *et al*. Liposome: classification, preparation, and applications. *Nanoscale Res. Lett*. **2013**. *8*, 102.

31. Suk, J. S.; Xu, Q.; Kim, N.; Hanes, J.; Ensign, L. M. PEGylation as a strategy for improving nanoparticle-based drug and gene delivery. *Adv. Drug Deliv. Rev*. **2016**. *99*, 28–51.

32. Barenholz, Y. Doxil® — The first FDA-approved nano-drug: Lessons learned. *J. Control. Release* **2012**. *160*, 117–134.

33. Li, S.-D.; Huang, L. Pharmacokinetics and Biodistribution of Nanoparticles. *Mol. Pharm.* **2008**. *5*, 496–504.

34. He, C.; Hu, Y.; Yin, L.; Tang, C.; Yin, C. Effects of particle size and surface charge on cellular uptake and biodistribution of polymeric nanoparticles. *Biomaterials* **2010**. *31*, 3657–3666.

35. Danhier, F. To exploit the tumor microenvironment: Since the EPR effect fails in the clinic, what is the future of nanomedicine? *J. Control. Release* **2016**. *244*, 108–121.

36. Kim, B.-K. *et al*. DOTAP/DOPE ratio and cell type determine transfection efficiency with DOTAP-liposomes. *Biochim. Biophys. Acta - Biomembr.* **2015**. *1848*, 1996–2001.

37. Friedman, A. D.; Claypool, S. E.; Liu, R. The smart targeting of nanoparticles. *Curr. Pharm. Des.* **2013**. *19*, 6315–29.

38. Farokhzad, O. C. *et al*. Nanoparticle-aptamer bioconjugates: a new approach for targeting prostate cancer cells. *Cancer Res.* **2004**. *64*, 7668–72.

39. Sun, Y. *et al*. Folic acid receptor-targeted human serum albumin nanoparticle formulation of cabazitaxel for tumor therapy. *Int. J. Nanomedicine* **2019**. *14*, 135–148.

40. Chen, Z. (Georgia). Small-molecule delivery by nanoparticles for anticancer therapy. *Trends Mol. Med.* **2010**. *16*, 594.

41. Kratschmer, C.; Levy, M. Effect of Chemical Modifications on Aptamer Stability in Serum. *Nucleic Acid Ther.* **2017**. *27*, 335–344.

42. Xiao, T. Innate immune recognition of nucleic acids. *Immunol. Res.* **2009**. *43*, 98–108.

43. Zhuang, J. *et al*. Targeted gene silencing in vivo by platelet membrane–coated metal-organic framework nanoparticles. *Sci. Adv.* **2020**. *6*, eaaz6108.

44. Buss, C. G.; Bhatia, S. N. Nanoparticle delivery of immunostimulatory oligonucleotides enhances response to checkpoint inhibitor therapeutics. *Proc. Natl. Acad. Sci.* **2020**. 01569.

45. Choi, J. *et al*. Nonviral polymeric nanoparticles for gene therapy in pediatric CNS malignancies. *Nanomedicine Nanotechnology, Biol. Med.* **2020**. *23*, 102115.

46. Tang, X. *et al*. Therapeutic Prospects of mRNA-Based Gene Therapy for Glioblastoma. *Front. Oncol.* **2019**. *9*, 1208

47. Ito, I. *et al*. Liposomal vector mediated delivery of the 3p FUS1 gene demonstrates potent antitumor activity against human lung cancer in vivo. *Cancer Gene Ther*. **2004**. *11*, 733–739.

48. Lu, C. *et al*. Phase I Clinical Trial of Systemically Administered TUSC2(FUS1)-Nanoparticles Mediating Functional Gene Transfer in Humans. *PLoS One* **2012**. *7*, e34833.

49. Yu, M.; Wu, J.; Shi, J.; Farokhzad, O. C. Nanotechnology for protein delivery: Overview and perspectives. *J. Control. Release* **2016**. *240*, 24–37.

50. Smith, S. A.; Selby, L. I.; Johnston, A. P. R.; Such, G. K. The Endosomal Escape of Nanoparticles: Toward More Efficient Cellular Delivery. *Bioconjug. Chem.* **2019**. *30*, 263–272.

51. Mirkin, C. A.; Letsinger, R. L.; Mucic, R. C.; Storhoff, J. J. A DNA-based method for rationally assembling nanoparticles into macroscopic materials. *Nature* **1996**. *382*, 607–609.

52. Cutler, J. I.; Zheng, D.; Xu, X.; Giljohann, D. A.; Mirkin, C. A. Polyvalent oligonucleotide iron oxide nanoparticle "click" conjugates. *Nano Lett*. **2010**. *10*, 1477–80.

53. Morris, W.; Briley, W. E.; Auyeung, E.; Cabezas, M. D.; Mirkin, C. A. Nucleic Acid–Metal Organic Framework (MOF) Nanoparticle Conjugates. *J. Am. Chem. Soc*. **2014**. *136*, 7261–7264.

54. Mitchell, G. P.; Mirkin, C. A.; Letsinger, R. L. Programmed Assembly of DNA Functionalized Quantum Dots. *J. Am. Chem. Soc*. **1999**. *121*, 8122–8123.

55. Young, K. L. *et al*. Hollow spherical nucleic acids for intracellular gene regulation based upon biocompatible silica shells. *Nano Lett*. **2012**. *12*, 3867–71.

56. Zhu, S.; Xing, H.; Gordiichuk, P.; Park, J.; Mirkin, C. A. PLGA Spherical Nucleic Acids. *Adv. Mater.* **2018**. *30*, 1707113.

57. Banga, R. J.; Chernyak, N.; Narayan, S. P.; Nguyen, S. T.; Mirkin, C. A. Liposomal spherical nucleic acids. *J. Am. Chem. Soc*. **2014**. *136*, 9866–9.

58. Brodin, J. D., Auyeung, E. & Mirkin, C. A. DNA-mediated engineering of multicomponent enzyme crystals. *Proc. Natl. Acad. Sci. U. S. A.* **2015**. *112*, 4564–9.

59. Banga, R. J. *et al*. Cross-Linked Micellar Spherical Nucleic Acids from Thermoresponsive Templates. *J. Am. Chem. Soc.* **2017**. *139*, 4278–4281.

60. Cutler, J. I. *et al*. Polyvalent Nucleic Acid Nanostructures. *J. Am. Chem. Soc.* **2011**. *133*, 9254–9257.

61. Lytton-Jean, A. K. R.; Mirkin, C. A. A Thermodynamic Investigation into the Binding Properties of DNA Functionalized Gold Nanoparticle Probes and Molecular Fluorophore Probes. *J. Am. Chem. Soc.* **2005**. *127*, 12754–12755.

62. Elghanian, R.; Storhoff, J. J.; Mucic, R. C.; Letsinger, R. L.; Mirkin, C. A. Selective colorimetric detection of polynucleotides based on the distance-dependent optical properties of gold nanoparticles. *Science* **1997**. *277*, 1078–81.

63. Park, S. Y.; Gibbs-Davis, J. M.; Nguyen, S. T.; Schatz, G. C. Sharp Melting in DNA-Linked Nanostructure Systems: Thermodynamic Models of DNA-Linked Polymers. *J. Phys. Chem. B* **2007**. *111*, 8785–8791.

64. Stepp, B. R.; Gibbs-Davis, J. M.; Koh, D. L. F.; Nguyen, S. T. Cooperative Melting in Caged Dimers of Rigid Small Molecule-DNA Hybrids. *J. Am. Chem. Soc.* **2008**. *130*, 9628–9629.

65. Rosi, N. L. *et al*. Oligonucleotide-modified gold nanoparticles for intracellular gene regulation. *Science* **2006**. *312*, 1027–30.

66. Patel, P. C. *et al*. Scavenger Receptors Mediate Cellular Uptake of Polyvalent Oligonucleotide-Functionalized Gold Nanoparticles. *Bioconjug Chem.* **2010**. *21*, 2250–2256.

67. Choi, C. H. J.; Hao, L.; Narayan, S. P.; Auyeung, E.; Mirkin, C. A. Mechanism for the endocytosis of spherical nucleic acid nanoparticle conjugates. *Proc. Natl. Acad. Sci.* **2013**. *110*, 7625–7630.

68. Seferos, D. S.; Prigodich, A. E.; Giljohann, D. A.; Patel, P. C.; Mirkin, C. A. Polyvalent DNA nanoparticle conjugates stabilize nucleic acids. *Nano Lett.* **2009**. *9*, 308–11.

69. Kawai, T.; Akira, S. The role of pattern-recognition receptors in innate immunity: update on Toll-like receptors. *Nat. Immunol.* **2010**. **11**, 373–384.

70. Massich, M. D.; Giljohann, D. A; Schmucker, A. L.; Patel, P. C.; Mirkin, C. A. Cellular response of polyvalent oligonucleotide-gold nanoparticle conjugates. *ACS Nano* **2010**. *4*, 5641–5646.

71. Storhoff, J. J; Elghanian, R; Mucic, R. C.; Mirkin, C. A.; Letsinger, R. L. One-Pot Colorimetric Differentiation of Polynucleotides with Single Base Imperfections Using Gold Nanoparticle Probes. *J. Am. Chem. Soc.* **1998**. *120*, 1959–1964.

72. Han, M. S.; Lytton-Jean, A. K. R.; Oh, B.-K.; Heo, J.; Mirkin, C. A. Colorimetric Screening of DNA-Binding Molecules with Gold Nanoparticle Probes. *Angew. Chemie Int. Ed.* **2006**. *45*, 1807–1810.

73. Taton, T. A.; Mirkin, C. A.; Letsinger, R. L. Scanometric DNA Array Detection with Nanoparticle Probes. *Science* **2000**. *289*, 1757-1760.

74. Nam, J.-M.; Thaxton, C. S.; Mirkin, C. A. Nanoparticle-based bio-bar codes for the ultrasensitive detection of proteins. *Science* **2003**. *301*, 1884–6.

75. Kim, D.; Daniel, W. L.; Mirkin, C. A. A Microarray-based Multiplexed Scanometric Immunoassay for Protein Cancer Markers Using Gold Nanoparticle Probes. *Anal. Chem.* **2009**. *81*, 9183-9187.

76. Alhasan, A. H. *et al*. Circulating microRNA signature for the diagnosis of very high-risk prostate cancer. *Proc. Natl. Acad. Sci.* **2016**. *113*, 10655-10660.

77. Seferos, D. S.; Giljohann, D. A.; Hill, H. D.; Prigodich, A. E.; Mirkin, C. A. Nano-Flares: Probes for Transfection and mRNA Detection in Living Cells. *J. Am. Chem. Soc.* **2007**. *129*, 15477–15479**.**

78. Prigodich, A. E. *et al*. Multiplexed nanoflares: mRNA detection in live cells. *Anal. Chem.* **2012**. *84*, 2062–6.

79. Halo, T. L. *et al*. NanoFlares for the detection, isolation, and culture of live tumor cells from human blood. *Proc. Natl. Acad. Sci.* **2014**. *111*, 17104–17109 .

80. Briley, W. E.; Bondy, M. H.; Randeria, P. S.; Dupper, T. J.; Mirkin, C. A. Quantification and real-time tracking of RNA in live cells using Sticky-flares. *Proc. Natl. Acad. Sci.* **2015**. *112*, 9591–9595.

81. Zheng, D.; Seferos, D. S.; Giljohann, D. A.; Patel, P. C.; Mirkin, C. A. Aptamer Nano-flares for Molecular Detection in Living Cells. *Nano Lett*. **2009**. *9*, 3258–3261.

82. Giljohann, D. A.; Seferos, D. S.; Prigodich, A. E.; Patel, P. C.; Mirkin, C. A. Gene Regulation with Polyvalent siRNA–Nanoparticle Conjugates. *J. Am. Chem. Soc*. **2009**. *131*, 2072–2073.

83. Yamankurt, G. *et al*. The effector mechanism of siRNA spherical nucleic acids. *Proc. Natl. Acad. Sci. U. S. A*. **2020**. *117*, 1312–1320.

84. Jensen, S. A. *et al*. Spherical nucleic acid nanoparticle conjugates as an RNAi-based therapy for glioblastoma. *Sci. Transl. Med*. **2013**. *5*, 209ra152.

85. Randeria, P. S. *et al*. siRNA-based spherical nucleic acids reverse impaired wound healing in diabetic mice by ganglioside GM3 synthase knockdown. *Proc. Natl. Acad. Sci*. **2015**. *112*, 5573–5578.

86. Wu, X. A., Choi, C. H. J., Zhang, C., Hao, L. & Mirkin, C. A. Intracellular fate of spherical nucleic acid nanoparticle conjugates. *J. Am. Chem. Soc*. **2014**. *136*, 7726–33.

87. Dasari, S.; Bernard Tchounwou, P. Cisplatin in cancer therapy: Molecular mechanisms of action. *Eur. J. Pharmacol*. **2014**. *740*, 364–378.

88. Dhar, S.; Daniel, W. L.; Giljohann, D. A.; Mirkin, C. A.; Lippard, S. J. Polyvalent Oligonucleotide Gold Nanoparticle Conjugates as Delivery Vehicles for Platinum(IV) Warheads. *J. Am. Chem. Soc*. **2009**. *131*, 14652–14653.

89. X, T. *et al*. Blurring the Role of Oligonucleotides: Spherical Nucleic Acids as a Drug Delivery Vehicle. *J. Am. Chem. Soc*. **2016.** *138*, 10834-10837.

90. Radovic-Moreno, A. F. *et al*. Immunomodulatory spherical nucleic acids. *Proc. Natl. Acad. Sci. U. S. A*. **2015**. *112*, 3892–7.

91. Yamankurt, G. *et al*. Exploration of the nanomedicine-design space with high-throughput screening and machine learning. *Nat. Biomed. Eng*. **2019**. *3*, 318–327.

92. Wang, S. *et al*. Rational vaccinology with spherical nucleic acids. *Proc. Natl. Acad. Sci. U. S. A*. **2019**. *116*, 10473–10481.

93. Guan, C. *et al*. RNA-Based Immunostimulatory Liposomal Spherical Nucleic Acids as Potent TLR7/8 Modulators. *Small* **2018**. *14*, e1803284.

94. Zhang, K.; Hao, L.; Hurst, S. J.; Mirkin, C. A. Antibody-linked spherical nucleic acids for cellular targeting. *J. Am. Chem. Soc*. **2012**. *134*, 16488–91.

95. Brodin, J. D., Sprangers, A. J., McMillan, J. R. & Mirkin, C. A. DNA-Mediated Cellular Delivery of Functional Enzymes. *J. Am. Chem. Soc*. **2015**. *137*, 14838–14841.

96. Speers, D. J. Clinical Applications of Molecular Biology for Infectious Diseases. *Clin. Biochem. Rev*. **2006**, *27*, 39–51.

97. Kamal, S. M.; Rashid, A. K. M. M.; Bakar, M. A.; Ahad, M. A. Anthrax: an update. *Asian Pac. J. Trop. Biomed*. **2011**. *1*, 496–501.

98. Kintzer, A. F. *et al*. The protective antigen component of anthrax toxin forms functional octameric complexes. *J. Mol. Biol*. **2009**. *392*, 614–29.

99. Singh, Y.; Klimpel, K. R.; Goel, S.; Swain, P. K.; Leppla, S. H. Oligomerization of anthrax toxin protective antigen and binding of lethal factor during endocytic uptake into mammalian cells. *Infect. Immun*. **1999**. *67*, 1853–9.

100. Kobiler, D. *et al*. Protective antigen as a correlative marker for anthrax in animal models. *Infect. Immun*. **2006**. *74*, 5871–6.

101. Mabry, R. *et al*. Detection of anthrax toxin in the serum of animals infected with Bacillus anthracis by using engineered immunoassays. *Clin. Vaccine Immunol*. **2006**. *13*, 671–7.

102. Morel, N. *et al*. Fast and sensitive detection of Bacillus anthracis spores by immunoassay. *Appl. Environ. Microbiol*. **2012**. *78*, 6491–8.

103. Dragan, A. I.; Albrecht, M. T.; Pavlovic, R.; Keane-Myers, A. M.; Geddes, C. D. Ultra-fast pg/ml anthrax toxin (protective antigen) detection assay based on microwave-accelerated metal-enhanced fluorescence. *Anal. Biochem*. **2012**. *425*, 54–61.

104. Tang, S. *et al*. Detection of anthrax toxin by an ultrasensitive immunoassay using europium nanoparticles. *Clin. Vaccine Immunol*. **2009**. *16*, 408–13.

105. Ghosh, N. *et al*. Detection of protective antigen, an anthrax specific toxin in human serum by using surface plasmon resonance. *Diagn. Microbiol. Infect. Dis*. **2013**. *77*, 14–19.

106. Oh, B. N. *et al*. Sensitive fluorescence assay of anthrax protective antigen with two new DNA aptamers and their binding properties. *Analyst* **2011**. *136*, 3384–8.

107. Slagle, K. M.; Ghosn, S. J. Immunoassays: Tools for Sensitive, Specific, and Accurate Test Results. *Lab. Med*. **1996**, *27*, 177–183.

108. Lequin, R. M. Enzyme Immunoassay (EIA)/Enzyme-Linked Immunosorbent Assay (ELISA). *Clin. Chem*. **2005**, *51*, 2415–2418.

109. Porstmann, T.; Kiessig, S. T. Enzyme Immunoassay Techniques. An Overview. *J. Immunol. Methods*, **1992**, *150*, 5–21.

110. Tang, S.; Hewlett, I. Nanoparticle-Based Immunoassays for Sensitive and Early Detection of HIV-1 Capsid (p24) Antigen. *J. Infect. Dis*. **2010**, *201*, S59–S64.

111. Díez-Buitrago, B.; Briz, N.; Liz-Marzán, L. M.; Pavlov, V. Biosensing Strategies Based on Enzymatic Reactions and Nanoparticles. *Analyst*, **2018**, *143*, 1727–1734.

112. Zheng, A.-X.; Li, J.; Wang, J.-R.; Song, X.-R.; Chen, G.-N.; Yang, H.-H. Enzyme-Free Signal Amplification in the DNAzyme Sensor Via Target-Catalyzed Hairpin Assembly. *Chem. Commun*. **2012**, *48*, 3112 -3114.

113. Farka, Z.; Juřík, T.; Kovář, D.; Trnková, L.; Skládal, P. Nanoparticle-Based Immunochemical Biosensors and Assays: Recent Advances and Challenges. *Chem. Rev*. **2017**, *117*, 9973–10042.

114. Rossi, N. L.; Mirkin, C. A. Nanostructures in Biodiagnostics. *Chem. Rev* **2005**, *105*, 1547-1562.

115. Kelley, S. O.; Mirkin, C. A.; Walt, D. R.; Ismagilov, R. F.; Toner, M.; Sargent, E. H. Advancing the Speed, Sensitivity and Accuracy of Biomolecular Detection using Multi-Length-Scale Engineering. *Nat. Nanotechnol*. **2014**, *9*, 969 –980.

116. El-Ansary, A.; Faddah, L. M. Nanoparticles as Biochemical Sensors. *Nanotechnol. Sci. Appl.* **2010**, *3*, 65–76.

117. Lee, J.-S.; Ulmann, P. A.; Han, M. S.; Mirkin, C. A. A DNA–Gold Nanoparticle-Based Colorimetric Competition Assay for the Detection of Cysteine. *Nano Lett.* **2008**, *8*, 529-533

118. Ambrosi, A.; Airò, F.; Merkoçi, A. Enhanced Gold Nanoparticle Based ELISA for a Breast Cancer Biomarker. *Anal. Chem.* **2010**, *82*, 1151–1156.

119. Gao, Z.; Xu, M.; Hou, L.; Chen, G.; Tang, D. Irregular-Shaped Platinum Nanoparticles as Peroxidase Mimics for Highly Efficient Colorimetric Immunoassay. *Anal. Chim. Acta*, **2013**, *776*, 79–86.

120. Chinen, A. B.; Guan, C. M.; Ferrer, J. R.; Barnaby, S. N.; Merkel, T. J.; Mirkin, C. A. Nanoparticle Probes for the Detection of Cancer Biomarkers, Cells, and Tissues by Fluorescence. *Chem. Rev.* **2015**, *115*, 10530–10574.

121. Geißler, D.; Charbonnière, L. J.; Ziessel, R. F.; Butlin, N. G.; Löhmannsröben, H.-G.; Hildebrandt, N. Quantum Dot Biosensors for Ultrasensitive Multiplexed Diagnostics. *Angew. Chemie Int. Ed.* **2010**, *49*, 1396–1401.

122. Bilan, R.; Ametzazurra, A; Brazhnik, K.; Escorza, S.; Fernández, D; Uríbarri, M; Nabiev, I; Alyona Sukhanova, A. Quantum-Dot-Based Suspension Microarray for Multiplex Detection of Lung Cancer Markers: Preclinical Validation and Comparison with the Luminex xMAP® System. *Sci. Rep.* **2017**, *7*, 44668.

123. Scott, A. W.; Garimella, V.; Calabrese, C. M.; Mirkin, C. A. Universal Biotin–PEG-Linked Gold Nanoparticle Probes for the Simultaneous Detection of Nucleic Acids and Proteins. *Bioconjug.*

124. Tian, D.; Duan, C.; Wang, W.; Cui, H. Ultrasensitive ElectroChemiluminescence Immunosensor Based on Luminol Functionalized Gold Nanoparticle Labeling. *Biosens. Bioelectron.* **2010**, *25*, 2290–2295.

125. Arya, S. K.; Estrela, P. Recent Advances in Enhancement Strategies for Electrochemical ELISA-Based Immunoassays for Cancer Biomarker Detection. *Sensors*. **2018**, *18*, 2010.

126. Ashley, M. J.; Bourgeois, M. R.; Murthy, R. R.; Laramy, C. R.; Ross, M. B.; Naik, R. R.; Schatz, G. C.; Mirkin, C. A. Shape and Size Control of Substrate-Grown Gold Nanoparticles for Surface-Enhanced Raman Spectroscopy Detection of Chemical Analytes. *J. Phys. Chem. C*, **2018**, *122*, 2307–2314.

127. Zhang, K.; Wang, Y.; Meiling Wu, M.; Liu, Y.; Shi, D.; Liu, B. On-Demand Quantitative SERS Bioassays Facilitated by Surface-Tethered Ratiometric Probes. *Chem. Sci*. **2018**, *9*, 8089–8093.

128. Li, Y.; Lu, Q.; Wu, S.; Wang, L; Shi, X. Hydrogen Peroxide Sensing Using Ultrathin Platinum-Coated Gold Nanoparticles with Core@Shell Structure. *Biosens. Bioelectron*. **2013**, *41*, 576–581.

129. Roy, R. K.; Njagi, J. I.; Farrell, B.; Halaciuga, I.; Lopez, M.; Goia, D.V. Deposition of Continuous Platinum Shells on Gold Nanoparticles by Chemical Precipitation. *J. Colloid Interface Sci*., **2012**, *369*, 91–95.

130. He, W. et al. Au@Pt Nanostructures as Oxidase and Peroxidase Mimetics for Use in Immunoassays. *Biomaterials*, **2011**, *32*, 1139–1147.

131. Hill, H. D.; Mirkin, C. A. The Bio-Barcode Assay for the Detection of Protein and Nucleic Acid Targets Using DTT-Induced Ligand Exchange. *Nat. Protoc*. **2006**, *1*, 324–336.

132. Zhao, J.; Wang, S; Lu, S.; Liu, G.; Jian Sun, J.; Yang, X. Fluorometric and Colorimetric Dual-Readout Immunoassay Based on an Alkaline Phosphatase-Triggered Reaction. *Anal. Chem*. **2019**, *91*, 7828–7834.

133. Zhang, R.; Li, N.; Sun, J.; Gao, F. Colorimetric and Phosphorimetric Dual-Signaling Strategy Mediated by Inner Filter Effect for Highly Sensitive Assay of Organophosphorus Pesticides. *J. Agric. Food Chem*. **2015**, *63*, 8947–8954.

134. Rudenko, N. V.; Abbasova, S. G.; Grishin, E. V. Preparation and Characterization of Monoclonal Antibodies to Bacillus Anthracis Protective Antigen. *Bioorg. Khim*. **2011**, *37*, 354–360.

135. Siddiqui, M. Z. Monoclonal Antibodies as Diagnostics; an Appraisal. *Indian J. Pharm. Sci.* **2010**, *72*, 12–17.

136. Savransky, V. et al. Pathology and Pathophysiology of Inhalational Anthrax in a Guinea Pig Model. *Infect. Immun.* **2013**, *81*, 1152–1163.

137. Frens, G. Controlled Nucleation for the Regulation of the Particle Size in Monodisperse Gold Suspensions. *Nat. Phys. Sci.* **1973**, *241*, 20–22.

138. Henry, J. P. Biological basis of the stress response. *Integr. Physiol. Behav. Sci.* **1992.** *27*, 66–83.

139. Juster, R.-P.; McEwen, B. S.; Lupien, S. J. Allostatic load biomarkers of chronic stress and impact on health and cognition. *Neurosci. Biobehav. Rev.* **2010**. *35*, 2–16.

140. Stegers-Jager, K. M.; Savas, M.; Waal, J.; Rossum, E. F. C.; Woltman, A. M. Gender-specific effects of raising Year-1 standards on medical students' academic performance and stress levels. *Med. Educ.* **2020**. *54*, 538–546.

141. Mauss, D.; Li, J.; Schmidt, B.; Angerer, P.; Jarczok, M. N. Measuring allostatic load in the workforce: a systematic review. *Ind. Health* **53**, 5 (2015).

142. Juster, R.-P. *et al*. A clinical allostatic load index is associated with burnout symptoms and hypocortisolemic profiles in healthy workers. *Psychoneuroendocrinology* **2011**. *36*, 797–805.

143. RM, S.; LM, R.; AU, M. How Do Glucocorticoids Influence Stress Responses? Integrating Permissive, Suppressive, Stimulatory, and Preparative Actions. *Endocr. Rev.* **2000**. *21*, 55-89.

144. Katsu, Y.; Iguchi, T. Cortisol. *Handb. Horm.* **2016**. *533-e95D-2*.

145. Kroboth, P. D.; Salek, F. S.; Pittenger, A. L.; Fabian, T. J.; Frye, R. F. DHEA and DHEA-S: A Review. *J. Clin. Pharmacol.* **1999**. *39*, 327–348.

146. Werner, M. *et al*. Preclinical challenges in steroid analysis of human samples. *J. Steroid Biochem. Mol. Biol.* **2010**. *121*, 505–512.

147. Stanczyk, F. Z.; Lee, J. S.; Santen, R. J. Standardization of Steroid Hormone Assays: Why, How, and When? *Cancer Epidemiol. Biomarkers & Prev.* **2007**. *16*, 1713–1719.

148. Krasowski, M. D. *et al*. Cross-reactivity of steroid hormone immunoassays: clinical significance and two-dimensional molecular similarity prediction. *BMC Clin. Pathol.* **2014**. *14*, 33.

149. Song, K.-M.; Lee, S.; Ban, C. Aptamers and their biological applications. *Sensors (Basel).* **2012**. *12*, 612–31.

150. Edwards, T. E.; Klein, D. J.; Ferré-D'Amaré, A. R. Riboswitches: small-molecule recognition by gene regulatory RNAs. *Curr. Opin. Struct. Biol.* **2007**. *17*, 273–279.

151. Darmostuk, M.; Rimpelova, S.; Gbelcova, H.; Ruml, T. Current approaches in SELEX: An update to aptamer selection technology. *Biotechnol. Adv.* **2015**. *33*, 1141–1161.

152. Godonoga, M. *et al*. A DNA aptamer recognising a malaria protein biomarker can function as part of a DNA origami assembly. *Sci. Rep.* **2016**. *6*, 21266.

153. Zimmermann, G. R.; Wick, C. L.; Shields, T. P.; Jenison, R. D.; Pardi, A. Molecular interactions and metal binding in the theophylline-binding core of an RNA aptamer. *RNA* **2000**. *6*, 659–67.

154. Yang, D. *et al*. Aptamer-based biosensors for detection of lead(ii) ion: a review. *Anal. Methods* **2017**. *9*, 1976–1990.

155. Xu, D. *et al*. Label-free electrochemical detection for aptamer-based array electrodes. *Anal. Chem.* **2005**. *77*, 5107–13.

156. MN, S.; P, de P.; DW, L. Aptamer-based Folding Fluorescent Sensor for Cocaine. *J. Am. Chem. Soc.* **2001**. *123*, 4928-4931.

157. W, Z.; W, C.; MA, B.; Y, L. Simple and Rapid Colorimetric Biosensors Based on DNA Aptamer and Noncrosslinking Gold Nanoparticle Aggregation. *Chembiochem* **2007**. *8*, 727-731.

158. Thiviyanathan, V.; Gorenstein, D. G. Aptamers and the next generation of diagnostic reagents. *Proteomics. Clin. Appl.* **2012**. *6*, 563–73.

159. Kalra, P.; Dhiman, A.; Cho, W. C.; Bruno, J. G.; Sharma, T. K. Simple Methods and Rational Design for Enhancing Aptamer Sensitivity and Specificity. *Front. Mol. Biosci.* **2018**. *5*, 41.

160. Yang, K.-A. *et al*. High-Affinity Nucleic-Acid-Based Receptors for Steroids. *ACS Chem. Biol*. **2017**. *12*, 3103–3112.

161. Neves, M. A. D., Slavkovic, S., Churcher, Z. R. & Johnson, P. E. Salt-mediated two-site ligand binding by the cocaine-binding aptamer. *Nucleic Acids Res*. **2016**. *45*, gkw1294.

162. Zhang, X., Servos, M. R. & Liu, J. Surface Science of DNA Adsorption onto Citrate-Capped Gold Nanoparticles. *Langmuir*. **2012**. *28*, 3896–3902.

163. Sander, J. D.; Joung, J. K. CRISPR-Cas systems for editing, regulating and targeting genomes. *Nat. Biotechnol*. **2014**. *32*, 347–355.

164. Karvelis, T. *et al*. crRNA and tracrRNA guide Cas9-mediated DNA interference in Streptococcus thermophilus. *RNA Biol*. **2013**. *10*, 841-851.

165. Jinek, M. et al. A programmable dual-RNA-guided DNA endonuclease in adaptive bacterial immunity. *Science*. **2012**. *337*, 816–821.

166. Anders, C.; Niewoehner, O.; Duerst, A.; Jinek, M. Structural basis of PAM-dependent target DNA recognition by the Cas9 endonuclease. *Nature*. **2014**. *513*, 569–573.

167. Fu, Y., Sander, J. D., Reyon, D., Cascio, V. M. & Joung, J. K. Improving CRISPR-Cas nuclease specificity using truncated guide RNAs. *Nat. Biotechnol*. **2014**. *32*, 279–284.

168. Cong, L. et al. Multiplex genome engineering using CRISPR/Cas systems. *Science*. **2013**. *339*, 819–23.

169. Durai, S. et al. Zinc finger nucleases: custom-designed molecular scissors for genome engineering of plant and mammalian cells. *Nucleic Acids Res*. **2005**. *33*, 5978–90.

170. Joung, J. K.; Sander, J. D. TALENs: a widely applicable technology for targeted genome editing. *Nat. Rev. Mol. Cell Biol*. **2012**. *14*, 49–55.

171. Paschon, D. E. et al. Diversifying the structure of zinc finger nucleases for high-precision genome editing. *Nat. Commun*. **2019**. *10*, 1133.

172. Moscou, M. J.; Bogdanove, A. J. A Simple Cipher Governs DNA Recognition by TAL Effectors. *Science*. **2009**. *326*, 1501.

173. Cermak, T. et al. Efficient design and assembly of custom TALEN and other TAL effector-based constructs for DNA targeting. *Nucleic Acids Res*. **2011**. *39*, e82.

174. Aparicio-Prat, E. et al. DECKO: Single-oligo, dual-CRISPR deletion of genomic elements including long non-coding RNAs. *BMC Genomics*. **2015**. *16*, 846.

175. Ran, F. A. et al. In vivo genome editing using Staphylococcus aureus Cas9. *Nature*. **2015**. *520*, 186–91.

176. Zetsche, B. et al. Cpf1 Is a Single RNA-Guided Endonuclease of a Class 2 CRISPR-Cas System. *Cell*. **2015**. *163*, 759–771.

177. Kim, D. et al. Genome-wide analysis reveals specificities of Cpf1 endonucleases in human cells. *Nat. Biotechnol*. **2016**. *34*, 863-868.

178. Hu, J. H. et al. Evolved Cas9 variants with broad PAM compatibility and high DNA specificity. *Nature*. **2018**. *556*, 57–63.

179. Kleinstiver, B. P. et al. High-fidelity CRISPR–Cas9 nucleases with no detectable genome-wide off-target effects. *Nature*. **2016**. *529*, 490–495.

180. Yang, L. et al. Genome-wide inactivation of porcine endogenous retroviruses (PERVs). *Science*. **2015**. *350*, 1101–1104.

181. Shalem, O. et al. Genome-scale CRISPR-Cas9 knockout screening in human cells. *Science*. **2014**. *343*, 84–7.

182. Hart, T. et al. High-Resolution CRISPR Screens Reveal Fitness Genes and Genotype-Specific Cancer Liabilities. *Cell*. **2015**. *163*, 1515–1526.

183. Chu, V. T. et al. Increasing the efficiency of homology-directed repair for CRISPR-Cas9-induced precise gene editing in mammalian cells. *Nat. Biotechnol*. **2015**. *33*, 543–548.

184. Chu, V. T. et al. Efficient generation of Rosa26 knock-in mice using CRISPR/Cas9 in C57BL/6 zygotes. *BMC Biotechnol*. **2016**. *16*, 4.

185. Dow, L. E. et al. Inducible in vivo genome editing with CRISPR-Cas9. *Nat. Biotechnol*. **2015**. *33*, 390–394.

186. Rees, H. A. & Liu, D. R. Base editing: precision chemistry on the genome and transcriptome of living cells. Nat. Rev. Genet. **2018**. *19*, 770-788.

187. Lee, H. K. et al. Targeting fidelity of adenine and cytosine base editors in mouse embryos. Nat. Commun. **2018**. *9*, 4804.

188. Yu, Y. et al. Cytosine base editors with minimized unguided DNA and RNA off-target events and high on-target activity. Nat. Commun. **2020**. *11*, 2052.

189. Anzalone, A. V. et al. Search-and-replace genome editing without double-strand breaks or donor DNA. *Nature*. **2019**. *576*, 149–157.

190. Gilbert, L. A. et al. CRISPR-Mediated Modular RNA-Guided Regulation of Transcription in Eukaryotes. *Cell*. **2013**. *154*, 442–451.

191. Zalatan, J. G. et al. Engineering Complex Synthetic Transcriptional Programs with CRISPR RNA Scaffolds. *Cell*. **2015**. *160*, 339-350.

192. Gilbert, L. A. et al. Genome-Scale CRISPR-Mediated Control of Gene Repression and Activation. *Cell*. **2014**. *159*, 647–61.

193. Chen, B. et al. Dynamic imaging of genomic loci in living human cells by an optimized CRISPR/Cas system. *Cell*. **2013**. *155*, 1479–1491.

194. McKenna, A. et al. Whole organism lineage tracing by combinatorial and cumulative genome editing. *Science*. **2016**. *42*, 237–241.

195. Giacca, M.; Zacchigna, S. Virus-mediated gene delivery for human gene therapy. *J. Control. Release* **2012**. *161*, 377–388.

196. Yin, H. et al. Genome editing with Cas9 in adult mice corrects a disease mutation and phenotype. *Nat. Biotechnol.* **2014**. *32*, 551–553.

197. Wang, W. et al. CCR5 Gene Disruption via Lentiviral Vectors Expressing Cas9 and Single Guided RNA Renders Cells Resistant to HIV-1 Infection. *PLoS One*. **2014**. *9*, e115987.

198. Doudna, J. A. et al. Genome editing. The new frontier of genome engineering with CRISPR-Cas9. *Science*. **2014**. *346*, 1258096.

199. Cho, S. W. et al. Analysis of off-target effects of CRISPR/Cas-derived RNA-guided endonucleases and nickases. *Genome Res*. **2014**. *24*, 132–41.

200. Hashimoto, M. et al. Electroporation enables the efficient mRNA delivery into the mouse zygotes and facilitates CRISPR/Cas9-based genome editing. *Sci. Rep*. **2015**. *5*, 11315.

201. Long, C. et al. Prevention of muscular dystrophy in mice by CRISPR/Cas9-mediated editing of germline DNA. *Science*. **2014**. *345*, 1184–1188.

202. Hendel, A. et al. Chemically modified guide RNAs enhance CRISPR-Cas genome editing in human primary cells. *Nat. Biotechnol*. **2015**. *33*, 985–989.

203. Kim, S.; Kim, D.; Cho, S. W.; Kim, J.; Kim, J.-S. Highly efficient RNA-guided genome editing in human cells via delivery of purified Cas9 ribonucleoproteins. *Genome Res*. **2014**. *24*, 1012–1019.

204. Zuris, J. A. et al. Cationic lipid-mediated delivery of proteins enables efficient protein-based genome editing in vitro and in vivo. *Nat. Biotechnol*. **2014**. *33*, 73–80.

205. Yang, Y. et al. A dual AAV system enables the Cas9-mediated correction of a metabolic liver disease in newborn mice. *Nat. Biotechnol*. **2016**. *34*, 334–338.

206. Yin, H. et al. Therapeutic genome editing by combined viral and non-viral delivery of CRISPR system components in vivo. *Nat. Biotechnol*. **2016**. *34*, 328–333.

207. Chen, S.; Lee, B.; Lee, A. Y.-F.; Modzelewski, A. J.; He, L. Highly Efficient Mouse Genome Editing by CRISPR Ribonucleoprotein Electroporation of Zygotes. *J. Biol. Chem*. **2016**. *291,* 14457-14467.

208. Suda, T.; Liu, D. Hydrodynamic Gene Delivery: Its Principles and Applications. *Mol. Ther.* **2007**. *15*, 2063–2069.

209. Daya, S.; Berns, K. I. Gene therapy using adeno-associated virus vectors. *Clin. Microbiol. Rev.* **2008**. *21*, 583–593.

210. Yin, H. et al. Non-viral vectors for gene-based therapy. *Nat. Rev. Genet.* **2014**. *15*, 541–555.

211. Cutler, J. I., Auyeung, E. & Mirkin, C. a. Spherical nucleic acids. *J. Am. Chem. Soc.* **2012**. *134*, 1376–1391.

212. Colletier, J.-P., Chaize, B., Winterhalter, M. & Fournier, D. Protein encapsulation in liposomes: efficiency depends on interactions between protein and phospholipid bilayer. *BMC Biotechnol.* **2002**. *2*, 9.

213. Rouge, J. L.; Hao, L.; Wu, X. A.; Briley, W. E.; Mirkin, C. A. Spherical Nucleic Acids as a Divergent Platform for Synthesizing RNA–Nanoparticle Conjugates through Enzymatic Ligation. *ACS Nano*. **2014**. *8*, 8837–8843.

214. Logisz, C. C. & Hovis, J. S. Effect of salt concentration on membrane lysis pressure. *Biochim. Biophys. Acta*. Biomembr. **2005**. *1717*, 104–108.

215. Cho, Y. W., Kim, J.-D. & Park, K. Polycation gene delivery systems: escape from endosomes to cytosol. *J. Pharm. Pharmacol.* **2003**. *55*, 721-734.

216. Lonez, C. et al. Fusogenic activity of cationic lipids and lipid shape distribution. *Cell. Mol. Life Sci.* **2010**. *67*, 483-94.

217. Huang, Y.; Ren, J.; Qu, X. Nanozymes: Classification, Catalytic Mechanisms, Activity Regulation, and Applications. *Chem. Rev.* **2019**, *119*, 4357–4412.

218. Wang, G.-L.; Jin, L.-Y.; Dong, Y.-M.; Wu, X.-M.; Li, Z.-J. Intrinsic enzyme mimicking activity of gold nanoclusters upon visible light triggering and its application for colorimetric trypsin detection. *Biosens. Bioelectron.* **2015**, *64*, 523–529.

219. Moayeri, M.; Wiggins, J. F.; Leppla, S. H. Anthrax Protective Antigen Cleavage and Clearance from the Blood of Mice and Rats. *Infect. Immun.* **2007**, *75*, 5175-5184.

220. de la Rica, R.; Stevens, M. M. Plasmonic ELISA for the ultrasensitive detection of disease biomarkers with the naked eye. *Nat. Nanotechnol.* **2012**. *7*, 821–824.

221. Chevalier, A. et al. Massively Parallel De Novo Protein Design for Targeted Therapeutics. *Nature*. **2017**, *550*, 74–79.

222. Munzar, J. D.; Ng, A.; Juncker, D. Comprehensive profiling of the ligand binding landscapes of duplexed aptamer families reveals widespread induced fit. *Nat*. *Commun*. **2018** *9*, 343.

223. Hurst, S. J.; Lytton-Jean, A. K. R.; Mirkin, C. A. Maximizing DNA Loading on a Range of Gold Nanoparticle Sizes. *Anal*. *Chem*. **2006**. *78*, 8313-8318.

224. Raff, H.; Raff, J. L.; Findling, J. W. Late-Night Salivary Cortisol as a Screening Test for Cushing's Syndrome 1. *J*. *Clin*. *Endocrinol*. *Metab*. **1998.** *83*, 2681–2686.

225. Vining, R. F.; McGinley, R. A.; Maksvytis, J. J.; Ho, K. Y. Salivary cortisol: a better measure of adrenal cortical function than serum cortisol. *Ann*. *Clin*. *Biochem*. **1983.** *20 (Pt 6)*, 329–35.

226. Whetzel, C. A.; Klein, L. C. Measuring DHEA-S in saliva: time of day differences and positive correlations between two different types of collection methods. BMC Res. Notes **2010.** *3*, 204.

227. Jorge Chavez lab, AFRL, unpublished.

228. Ebrahimi, S. B.; Samanta, D.; Cheng, H. F.; Nathan, L. I.; Mirkin, C. A. Forced Intercalation (FIT)-Aptamers. *J*. *Am*. *Chem*. *Soc*. **2019**. *141*, 13744–13748.

229. Köhler, O.; Jarikote, D. V.; Seitz, O. Forced Intercalation Probes (FIT Probes): Thiazole Orange as a Fluorescent Base in Peptide Nucleic Acids for Homogeneous Single-Nucleotide-Polymorphism Detection. *ChemBioChem*. **2005** *6*, 69–77.

230. Kamiyama, D. et al. Versatile protein tagging in cells with split fluorescent protein. *Nat*. *Commun*. **2016.** *7*, 11046.

231. Feng, S. *et al*. Improved split fluorescent proteins for endogenous protein labeling. *Nat*. *Commun*. **2017**. *8*, 370.

232. Lönn, P. et al. Enhancing Endosomal Escape for Intracellular Delivery of Macromolecular Biologic Therapeutics. *Sci*. *Rep*. **2016**. *6*, 32301.

233. Skakuj, K. et al. Conjugation Chemistry-Dependent T-Cell Activation with Spherical Nucleic Acids. *J*. *Am*. *Chem*. *Soc*. **2018**. *140*, 1227–1230.

APPENDIX A: Supplementary Tables for Chapter 3

Table A1. Oligonucleotide sequences for PEG backfilling experiments.

Name	Sequence
DIS11th_3T DHEA-S	5'-GGA CGT GGA TTT TCC GCA TAC GAA
Aptamer	GTT GTC C AAA AAA A-SH-3'
8bp DIS11th_3T flare	5'-Cy5-GGA CAA CT-3'
	5'-GACAA GGAAA ATCCT TCAAC GAAGT
MN19 cocaine aptamer	GGGTC AAA AAA A-SH-3'
8bp MN19 flare	5'-Cy5-GAC CCA CT-3'

Table A2. Aptamers tested in the microarray experiments.

Name	Target	Sequence
10_51	DHEA-S	Cy3-CTCTCGGGACGACGCCAGAAGTTTACGAGGATATGGTAACATAGTCGTCCC
15-1	DHEA-S	Cy3-GAATGGATATGGGCAATGCGGGGTGGAGAATGGTTGCCGCACTTCGGC
15-3	None	Cy3-GAATGGATGAGGGTTGGAAGGGAGGGGCCCGGGGTGGGCCATCGTTCG
CSS.1	DHEA-S/ Cortisol	Cy3-CTCACGACGCCCGCATGTTCCATGGATAGTCTTGACTAGTCGT
DCA6th_23	DHEA-S	Cy3-GGCTCTCGGGACGACaaGGATTTTCCtagaACGAAGTtgGTCGTCCC
DIS11th_3	DHEA-S	Cy3-GGCTCTCGGGACGtGGATTTTCCgcatACGAAGTtGTCCC
FMN	Riboflavin	Cy5-GGCGUGUAGGAUAUGCUUCGGCAGAAGGACACGCC
THY	Theophylline	Cy5-GGUGAUACCAGCAUCGUCUUGAUGCCCUUGGCAGCACC
TNT	TNT	Cy5-GUCUAGACUGCAGAGUUAGUGGCCGGUGUCUGUAUGAGUCGAGUUUUGCAUUUCUGCAGGUCGAC
Dopa2	Dopamine	Cy5-GGGAAUUCCGCGUGUGCGCCGCGGAAGAGGGAAUAUAGAGGCCAGCACAUAGUGAGGCCCUCCUCCC

APPENDIX B: Supplementary Tables for Chapter 4

Table A1: Sequences for CRISPR SNA hybridization studies.

Name	Sequence
Hyb_SNA_ F-Actin	5'-HS-(C3H6)-AAA AAA TCC TAC TAT CGC TCG CT-3'
Hyb_SNA_ Scrambled_ Actin	5'-HS-(C3H6)-AAA AAA CCA TTG CGA CCC CGC CT-3'
Hyb_sgRNA _Act_AAVS 1_1 IVT Template	5'-TGGCTAATACGACTCACTATAGGGAGAGTCACCAATCCTGTCCCTAGGTTTTAGAG CTATGAAAATAGCAAGTTAAAATAAGGCTAGTCCGTTATCAACTTGAAAAAGTGGCA CCGAGT CGGTGCTT-*ACGGCTCCGGCA*-3'
Hyb_sgRNA _Act_AAVS 1_1 sgRNA	5'-**GUCACCAAUCCUGUCCCUAG-** GUUUUAGAGCUAUGAAAAUAGCAAGUUAAAAUAAGGCUAGUCCGUUAUCAACUU GAAAAAGUGGCACCGAGUCGGUGCUUGAGAUGCUA-*ACGGCUCCGGCA*-3'
Hyb_sgRNA _Scr_eGFP IVT Template	5'-TGGCTAATACGACTCACTATAGGGAGAGAGCTGGACGGCGACGTAAAGTTTTAGA GCTATGAAAATAGCAAGTTAAAATAAGGCTAGTCCGTTATCAACTTGAAAAAGTGGC ACCGAGTCGGTGCTT-GAGATGCTAACG-'3
Hyb_sgRNA _Scr_eGFP sgRNA	5'-**GAGCUGGACGGCGACGUAAA-** GUUUUAGAGCUAUGAAAAUAGCAAGUUAAAAUAAGGCUAGUCCGUUAUCAACUU GAAAAAGUGGCACCGAGUCGGUGCUUGAGAUGCUA-*GAGATGCTAACG*-3'
Hyb_sgRNA _Act_eGFP IVT Template	5'-TGGCTAATACGACTCACTATAGGGAGAGAGCTGGACGGCGACGTAAAGTTTTAGA GCTATGAAAATAGCAAGTTAAAATAAGGCTAGTCCGTTATCAACTTGAAAAAGTGGC ACCGAGTCGGTGCTT-*ACGGCTCCGGCA*-3'

Hyb_sgRNA 5'-**GAGCUGGACGGCGACGUAAA-**

_Act_eGFP GUUUUAGAGCUAUGAAAAUAGCAAGUUAAAAUAAGGCUAGUCCGUUAUCAACUU

_1 sgRNA GAAAAAGUGGCACCGAGUCGGUGCUUGAGAUGCUA-*ACGGCUCCGGCA*-3'

Table A2: Sequences for liposomal CRISPR SNA studies. **Red** = 2'O-methyl RNA base, * = phosphorothioate backbone between RNA bases.

Name	Sequence
DBCO-Cy3-DNA	5'-DBCO-TTTTTTTTTT-Cy3-AATTCACCCAAC-3'
GFP crRNA	5'-G*A*GCUGGACGGCGACGUAAAGUUUUAGAGCUAUG*C*U-3'
EMX1 crRNA	5'-G*A*GUCCGAGCAGAAGAAGAAGUUUUAGAGCUAUG*C*U-3'
FANCF crRNA	5'-G*G*AACCCUUCUGCAGCACCGUUUUAGAGCUAUGC*U-3'
tracrRNA	5'-A*G*CAUAGCAAGUUAAAAUAAGGCUAGUCCGUUAUCAACU UGAAAAAGUGGCACCGAGUCGGUGCUU*U*-3'

Table A3: Primers for CRISPR SNA gene editing studies.

Name	Sequence
EMX1_HTS_F	5'-ACA CTC TTT CCC TAC ACG ACG CTC TTC CGA TCT ACA TCG CAG CTC AGC CTG AGT GTT GA-3'
EMX1_HTS_R	5'-TGG AGT TCA GAC GTG TGC TCT TCC GAT CTC TCG TGG GTT TGT GGT TGC-3'
FANCF_HTS_F	5'-ACA CTC TTT CCC TAC ACG ACG CTC TTC CGA TCT TGG TCA CAT TGC AGA GAG GCG TAT CA-3'
FANCF_HTS_R	5'-TGG AGT TCA GAC GTG TGC TCT TCC GAT CTG GGG TCC CAG GTG CTG AC-3'

APPENDIX C: Supplementary Code for Chapter 3

Code C1. Conformational Selection Equilibrium Model Markdown File

```python
#import required packages/functions: numpy, fsolve, matplotlib.pyplot
import numpy as np
from scipy.optimize import fsolve
import matplotlib.pyplot as plt
get_ipython().magic(u'matplotlib inline') #Enables jupyter to plot matplotlib things in-line
```

```python
#Set the values for total Aptamer + Flare strand concentration, and a range of DHEA-S concentrations
#Okay. Let's try a range of parameter changes.
#Let's try for 0.1 nM NP, 1 nM NP, 10 nM NP, and 100 nM NP.
#Let's also try for Kd_Hyb = 0.1 nM, 1 nM, 10 nM, and 100 nM
#Let's also try for Kd_Apt = 0.01 nM, 0.1 nM, 1 nM, 10 nM, 100 nM, 1 μM, 10 μM, and 100 μM

#First, 10 nM Aptamer NanoFlare
A_totalConc = 300   #Total Aptamer concentration, in nM
F_totalConc = 37.5   #Total Flare strand, in nM
numDataPoints = 100 #This variable dictates how many data points to consider and plot

D_concArray = np.zeros(shape=(numDataPoints,1))
D_concArray[0] = 1 #Smallest concentration we'll try is 1 nM.

D_increment = (1e9)**(1/float(numDataPoints)) #This will set a geometrically even increment between data points on a log axis plot.

DHEASconc = 1 #This concentration (in nM) value will incrementally increase as the FOR loop iterates

for i in range(1,numDataPoints-1):
    DHEASconc = DHEASconc * D_increment
    D_concArray[i] = DHEASconc

for i in range(0, numDataPoints-1):
    if D_concArray[i] == 0:
        D_concArray[i] = np.NaN
```

```python
#Set the Kd_Apt and Kd_Hyb values.
Kd_Apt = 27000  #Experimentally determined value of Kd_Apt, in nM
Kd_Hyb = 11    #Experimentally estimated value of Kd_Hyb, in nM
```

```
#Set the initial concentration of free flare strand. This may not be necessary
if I get rid of DeltaF_Rel.
F_0 = .1
```
```python
#Now define arrays of values for A, D, F, AD, AF, Kd_Det, and DeltaF_Rel
A_array = np.zeros(shape=(numDataPoints,1))
F_array = np.zeros(shape=(numDataPoints,1))
D_array = np.zeros(shape=(numDataPoints,1))
AF_array = np.zeros(shape=(numDataPoints,1))
AD_array = np.zeros(shape=(numDataPoints,1))
Kd_DetArray = np.zeros(shape=(numDataPoints,1))
DeltaF_RelArray = np.zeros(shape=(numDataPoints,1))
DeltaF_Tricky_RelArray = np.zeros(shape=(numDataPoints,1))
```
```python
#Now set up a FOR loop to generate values in all these arrays based on changing
DHEA-S concentrations

for i in range(0,numDataPoints-1):
    #Define all the equations here, in X*Y+C^2-D format. All equations should
= 0.
    def equations(p): #defines a dummy variable, tells which variables are
working
        A, D, F, AD, AF = p
        return (abs(A) + abs(AD) + abs(AF) - A_totalConc,
                abs(D) + abs(AD) - D_concArray[i],
                abs(F) + abs(AF) - F_totalConc,
                abs(D)*abs(A)/abs(AD) - Kd_Apt,
                abs(F)*abs(A)/abs(AF) - Kd_Hyb)
    #Use fsolve to computaionally solve the equations for a given set of input
parameters.
    #Note: Something's wrong with DeltaF_Rel; only spits out the value 1.
Unclear if it's even necessary.
    A, D, F, AD, AF = fsolve(equations,
                             (A_totalConc,
                             D_concArray[i],
                             F_totalConc,
                             A_totalConc,
                             F_totalConc,
                             ))
    A_array[i] = A
    F_array[i] = F
    D_array[i] = D
    AF_array[i] = AF
    AD_array[i] = AD
    Kd_DetArray[i] = (abs(AF)+abs(A))*abs(D)/abs(AD)
```

```
for i in range(0,numDataPoints-1):
    DeltaF_RelArray[i] = F_array[i] / F_array[0]
```
```python
#Now, 'absolutize' all the arrays. Sometimes fsolve converges to -1 * the correct variable.
cleanA = np.absolute(A_array)
cleanF = np.absolute(F_array)
cleanD = np.absolute(D_array)
cleanAF = np.absolute(AF_array)
cleanAD = np.absolute(AD_array)
cleanKd_Det = np.absolute(Kd_DetArray)
cleanDeltaF_Rel = np.absolute(DeltaF_RelArray)
```
```python
#Sometimes fsolve spits out obviously wrong answers, which show up as outliers.
#Make a function cleanModels that removes array values that deviate too drastically from their immediate neighbors.
#Works for a vertical 1D array
def cleanModels(array):
    arraySize = array.shape[0]
    cleanArray = np.zeros(shape=(arraySize,1))
    delta = np.zeros(shape=(arraySize,1)) #make an array for recording the change in value from one element to the next
#    for i in range(0,array.shape[0]-1):
#        delta[i] = abs(array[i] - array[i+1])

#Get rid of the very last data point, which can be an outlier
    array[arraySize-1] = np.NaN

#If I could be a little cleverer about defining delta I bet all of this would go away
#Define delta based on the median of the nearest five data points
#This generates an array of Delta values (the difference between one point and the five points around it)
#Perhaps the best way to spot and eliminate outliers is to compare the median Delta to the average delta
#The trouble with this approach is that it has difficulty removing many outliers in a row
    for i in range(0,4):
        delta[i] = abs(array[i] - np.median(array[i:i+5]))
    for i in range(5, array.shape[0]-2):
        delta[i] = abs(array[i] - np.median(array[i-2:i+2]))
    for i in range(array.shape[0]-2, array.shape[0]-1):
        delta[i] = abs(array[i] - np.median(array[i-5:i]))

    #Make medianDelta into an array, segment into chunks of 10 data points
    medianDelta = np.zeros(shape=(array.shape[0],1))
```

```
    for i in range(0, array.shape[0]-11):
        medianDelta[i] = np.median(delta[i:i+10])

    for i in range(1,array.shape[0]):
        if delta[i] > 10*(medianDelta[i]):
            cleanArray[i] = np.NaN
        else:
            cleanArray[i] = array[i]

    for i in range(1,array.shape[0]):
        if array[i] > 10*np.average(array):
            cleanArray[i] = np.NaN
        else:
            cleanArray[i] = array[i]
    return cleanArray
```

```python
#Now, clean up all the arrays of values!
cleanA = cleanModels(cleanA)
cleanF = cleanModels(cleanF)
cleanD = cleanModels(cleanD)
cleanAF = cleanModels(cleanAF)
cleanAD = cleanModels(cleanAD)
cleanKd_Det = cleanModels(cleanKd_Det)
cleanDeltaF_Rel = cleanModels(cleanDeltaF_Rel)

#Now, try to delete the first values of all the arrays to get rid of that
annoying first value that shows up as zero
D_concArray[0] = np.NaN
#Hey, it worked!
```

```python
#Now, plot the cleaned up data as a function of [DHEA-S]!
plt.figure(1)
plt.semilogx(D_concArray, cleanDeltaF_Rel, 'ro')
plt.xlabel('[DHEA-S], nM')
plt.ylabel("Relative Delta [F]")
plt.show()

plt.figure(2)
plt.semilogx(D_concArray, cleanF, 'bo')
plt.xlabel('[DHEA-S], nM')
plt.ylabel('[Free Flare], nM')
plt.show()

plt.figure(3)
plt.semilogx(D_concArray, cleanA, 'go')
```

```
plt.xlabel('[DHEA-S], nM')
plt.ylabel('[Free Aptamer], nM')
plt.show()

plt.figure(4)
plt.semilogx(D_concArray, cleanAF, 'ro')
plt.xlabel('[DHEA-S], nM')
plt.ylabel('[Aptamer-Flare complex], nM')
plt.show()

plt.figure(5)
plt.semilogx(D_concArray, cleanAD, 'bo')
plt.xlabel('[DHEA-S], nM')
plt.ylabel('[Aptamer-DHEAS complex], nM')
plt.show()

plt.figure(6)
plt.semilogx(D_concArray, cleanKd_Det, 'bo')
plt.xlabel('[DHEA-S], nM')
plt.ylabel('[Kd_Det], nM')
plt.show()
```

```python
#Export the relative fluorescence change data to a csv

import csv

csvfile = "filename.csv"
with open(csvfile, "w") as output:
    writer = csv.writer(output, lineterminator='\n')
    for val in cleanDeltaF_Rel:
        writer.writerow(val)
```

```python
#Okay. Now, I've got a sense of how cranky the equations solver function is.
What I would like to do now is test out
#whether I can get the F_relative fluorescence to work based
#on the equation I derived
```

```python
#Now define arrays of values for A, D, F, AD, AF, Kd_Det, and DeltaF_Rel
A_array = np.zeros(shape=(numDataPoints,1))
F_array = np.zeros(shape=(numDataPoints,1))
D_array = np.zeros(shape=(numDataPoints,1))
AF_array = np.zeros(shape=(numDataPoints,1))
AD_array = np.zeros(shape=(numDataPoints,1))
Kd_DetArray = np.zeros(shape=(numDataPoints,1))
```

```
DeltaF_RelArray = np.zeros(shape=(numDataPoints,1))
DeltaF_derived_RelArray = np.zeros(shape=(numDataPoints,1))
```

```python
#Now set up a FOR loop to generate values in all these arrays based on changing DHEA-S concentrations
for i in range(0,numDataPoints-1):
    #Define all the equations here, in X*Y+C^2-D format. All equations should = 0.
    def equations(p): #I think this defines a dummy variable, tells which variables are working
        A, D, F, AD, AF = p
        return (abs(A) + abs(AD) + abs(AF) - A_totalConc,
                abs(D) + abs(AD) - D_concArray[i],
                abs(F) + abs(AF) - F_totalConc,
                abs(D)*abs(A)/abs(AD) - Kd_Apt,
                abs(F)*abs(A)/abs(AF) - Kd_Hyb)


    #Use fsolve to computationally solve the equations for a given set of input parameters.
    #Note: Something's wrong with DeltaF_Rel; only spits out the value 1. Unclear if it's even necessary.
    A, D, F, AD, AF = fsolve(equations,
                             (A_totalConc,
                             D_concArray[i],
                             F_totalConc,
                             A_totalConc,
                             F_totalConc,
                             ))
    A_array[i] = A
    F_array[i] = F
    D_array[i] = D
    AF_array[i] = AF
    AD_array[i] = AD
    Kd_DetArray[i] = (abs(AF)+abs(A))*abs(D)/abs(AD)
    DeltaF_RelArray[i] = F_array[i] / F_array[1]
    DeltaF_derived_RelArray[i] = 1 + (abs(A)*abs(D)/(Kd_Apt*abs(AD)))
```

```python
#Now, just do the cleaning and stuff for DeltaF_RelArray and DeltaF_derived_RelArray

cleanDeltaF_Rel = np.absolute(DeltaF_RelArray)
```

```
cleanDeltaF_derived_RelArray = np.absolute(DeltaF_derived_RelArray)

cleanDeltaF_Rel = cleanModels(cleanDeltaF_Rel)
cleanDeltaF_derived_RelArray = cleanModels(cleanDeltaF_derived_RelArray)
```
```python
#Now, plot the cleaned up data as a function of [DHEA-S]!
plt.figure(1)
plt.semilogx(D_concArray, cleanDeltaF_Rel, 'ro')
plt.xlabel('[DHEA-S], nM')
plt.ylabel("Relative Delta [F]")
plt.show()

#Now, plot the cleaned up data as a function of [DHEA-S]!
plt.figure(1)
plt.semilogx(D_concArray, cleanDeltaF_derived_RelArray, 'bo')
plt.xlabel('[DHEA-S], nM')
plt.ylabel("Derived Relative Delta [F]")
plt.show()

#Okay. So the derived Relative Delta F Doesn't actually give a
#meaningful number. It's literally just 1 + 1.
```
```python
#Now, time to plot data recapitulating Mirau lab's
#aptamer quenching experiments.
#In this case, the species that decreased over time is [A],
#While the species that increases over time is [AD]
#(with increasing concentrations of flare strand)

F_totalConc = 200.0   #Total Flare Strand concentration, in nM

A_concArray = np.zeros(shape=(10,1)) #Total Aptamer-BHQ conc, in nM

A_concArray[0] = 10.0
A_concArray[1] = 14.0
A_concArray[2] = 30.0
A_concArray[3] = 40.0
A_concArray[4] = 80.0
A_concArray[5] = 200.0
A_concArray[6] = 400.0
A_concArray[7] = 700.0
A_concArray[8] = 1200.0
A_concArray[9] = 2000.0

#In this case, the concentration of D is always 0.
D_totalConc = 0
```

```
#numDataPoints = 100 #This variable dictates how many data points to consider
and plot
#D_concArray = np.zeros(shape=(numDataPoints,1))
#D_concArray[0] = 100 #Smallest concentration we'll try is 1 nM.

#D_increment = (1e5)**(1/float(numDataPoints)) #This will set a geometrically
even increment between data points on a log axis plot.

#DHEASconc = 100 #This concentration (in μM) value will incrementally increase
as the FOR loop iterates

#for i in range(1,numDataPoints-1):
#    DHEASconc = DHEASconc * D_increment
#    D_concArray[i] = DHEASconc
```
```python
#Now define arrays of values for A, D, F, AD, AF, Kd_Det, and DeltaF_Rel
A_array = np.zeros(shape=(10,1))
F_array = np.zeros(shape=(10,1))
AF_array = np.zeros(shape=(10,1))
DeltaF_RelArray = np.zeros(shape=(10,1))

#Set the Kd_Hyb value.
Kd_Hyb = 100    #Experimentally estimated value of Kd_Hyb, in nM

```
```python
#Now set up a FOR loop to generate values in all these arrays based on changing
DHEA-S concentrations

for i in range(0,9):
    #Define all the equations here, in X*Y+C^2-D format. All equations should
= 0.
    def equations(p): #I think this defines a dummy variable, tells which
variables are working
        A, F, AF = p
        return (abs(A) + abs(AF) - A_concArray[i],
                abs(F) + abs(AF) - F_totalConc,
                abs(F)*abs(A)/abs(AF) - Kd_Hyb)
    #Use fsolve to computationally solve the equations for a given set of input
parameters.

    A, F, AF = fsolve(equations,(50, F_totalConc, 50))
    A_array[i] = A
    F_array[i] = F
    AF_array[i] = AF
    DeltaF_RelArray[i] = F_array[i] / F_array[0]
```

```
#Now, 'absolutize' all the arrays. Sometimes fsolve converges to -1 * the
correct variable.
cleanA = np.absolute(A_array)
cleanF = np.absolute(F_array)
cleanAF = np.absolute(AF_array)
cleanDeltaF_Rel = np.absolute(DeltaF_RelArray)

```
```python
#Now, plot the cleaned up data as a function of [DHEA-S]!
plt.figure(1)
plt.semilogx(A_concArray, cleanDeltaF_Rel, 'ro')
plt.xlabel('[Aptamer-BHQ], nM')
plt.ylabel("Relative Delta [F]")
plt.show()

#Now, plot the cleaned up data as a function of [DHEA-S]!
plt.figure(1)
plt.semilogx(A_concArray, cleanF, 'bo')
plt.xlabel('[Aptamer-BHQ], nM')
plt.ylabel("[F], nM")
plt.show()

#Now, plot the cleaned up data as a function of [DHEA-S]!
plt.figure(1)
plt.semilogx(A_concArray, cleanAF, 'go')
plt.xlabel('[Aptamer-BHQ], nM')
plt.ylabel("[AF], nM")
plt.show()
```

```python
#Export the relative fluorescence change data to a csv

import csv

csvfile = "20180203_FlareBindingModelPredictions_KdHyb_100nM.csv"

with open(csvfile, "w") as output:
    writer = csv.writer(output, lineterminator='\n')
    for val in cleanDeltaF_Rel:
        writer.writerow(val)
```

Code C2. Induced fit and conformational selection kinetic models markdown file.

```python
#Import all the relevant packages

#PySB, which runs the model
from pysb import *

#ODE Solvers
from pysb.integrate import odesolve
from pysb.simulator import ScipyOdeSimulator
#import cython
#^at some point, learn to run the ODEs off cython, which is faster than python

#Plotting packages
import numpy as np
import pylab as pl
from pylab import plot, linspace
import pygraphviz

#For exporting network and simulation results to CSV/Excel/py/dot/PDF files
from pysb.export import export
import pandas as pd
```

```python
!pip freeze > requirements.txt
```

```python
#now, build the two models: conformational selection and induced fit.
#Instead of storing the model code in different .py files, make functions that will generate/return the models
#This way, the model parameters can be changed more easily
#First, Conformational Selection model

#This time, define k_HybF and k_HybR using the experimentally determined parameters

#BLI experimentally determined k_HybF: 18,000 1/(M*s) * 1 M / 1,000,000 μM = .018
#BLI experimentally determined k_HybR: 0.00555 1/s

def genAFCSmodel(k_HybF=0.018,
                 k_HybR=0.00555,
```

```
                k_AptF=0.1,
                k_AptR=3,
                AF_0=1,
                D_0=1000,
                A_unb_0=0.01,
                F_unhyb_0=0.01,
                AD_0=0):
    #define the Model object that the function will return
    a = Model()

    #Define the molecules in the model, and their associated parameters/binding
sites
    Monomer('A', ['sf', 'sd'])
    Monomer('F', ['sf'])
    Monomer('D', ['sd'])

    #Define the reaction rate parameters required for the model
    Parameter('kHybF', k_HybF)
    Parameter('kHybR', k_HybR)
    Parameter('kAptF', k_AptF)
    Parameter('kAptR', k_AptR)

    #Define the initial concentration parameters required for the model
    Parameter('AF_init', AF_0)
    Parameter('D_init', D_0)
    Parameter('A_unb_init', A_unb_0)
    Parameter('F_unhyb_init', F_unhyb_0)
    Parameter('AD_init', AD_0)

    #Define the initial molecular species and concentrations at the start of
the model simulation
    Initial(A(sf=1, sd=None) % F(sf=1), AF_init)
    Initial(D(sd=None), D_init)
    Initial(A(sf=None, sd=None), A_unb_init)
    Initial(F(sf=None), F_unhyb_init)
    Initial(A(sf=None, sd=1) % D(sd=1), AD_init)

    #Define the reactions the molecules in the model can undergo, with their
associated rates
    #Reaction 1: A + F <> AF
    Rule('A_binds_F', A(sf=None, sd=None) + F(sf=None) | A(sf=1, sd=None) %
F(sf=1), kHybF, kHybR)
    #Reaction 2: A + D <> AD
    Rule('A_binds_D', A(sf=None, sd=None) + D(sd=None) | A(sf=None, sd=1) %
D(sd=1), kAptF, kAptR)

    #Define all the molecular species you want to be able to call, return, and
plot after simulating
```

```
    Observable('AF', A(sf=1, sd=None) % F(sf=1))
    Observable('F_unhyb', F(sf=None))
    Observable('D_unb', D(sd=None))
    Observable('A_unb', A(sf=None, sd=None))
    Observable('AD', A(sf=None, sd=1) % D(sd=1))

    #Return the model
    return a
```

```python
#And now, the induced fit model

#Now create a method to generate an IF model with specified parameters
#Set default values here
def genAFIFmodel(k_HybF=0.018,
                 k_HybR=0.00555,
                 k_AptF=0.1,
                 k_AptR=3,
                 k_AptIF_F=0.001,
                 k_AptIF_R=10,
                 k_HybIF_F=0.01,
                 k_HybIF_R=0.01,
                 AF_0=1,
                 D_0=1000,   #100 μM DHEA-S
                 A_unb_0=0.01,
                 F_unhyb_0=0.01,
                 AD_0=0,
                 ADF_0=0):
    #Define the Model object that the function will return
    a = Model()

    #Define the molecules in the model, and their associated parameters/binding
sites
    Monomer('A', ['sf', 'sd'])
    Monomer('F', ['sf'])
    Monomer('D', ['sd'])

    #Define the reaction rate parameters required for the model
    Parameter('kHybF', k_HybF)
    Parameter('kHybR', k_HybR)
    Parameter('kAptF', k_AptF)
    Parameter('kAptR', k_AptR)
    Parameter('kAptIF_F', k_AptIF_F)
    Parameter('kAptIF_R', k_AptIF_R)
    Parameter('kHybIF_F', k_HybIF_F)
```

```
    Parameter('kHybIF_R', k_HybIF_R)
    Parameter('AF_init', AF_0)

    #Define the initial concentration parameters required for the model
    Parameter('D_init', D_0)
    Parameter('A_unb_init', A_unb_0)
    Parameter('F_unhyb_init', F_unhyb_0)
    Parameter('AD_init', AD_0)
    Parameter('ADF_init', ADF_0)

    #Define the initial molecular species and concentrations at the start of
the model simulation
    Initial(A(sf=1, sd=None) % F(sf=1), AF_init)
    Initial(D(sd=None), D_init)
    Initial(A(sf=None, sd=None), A_unb_init)
    Initial(F(sf=None), F_unhyb_init)
    Initial(A(sf=None, sd=2) % D(sd=2), AD_init)
    Initial(A(sf=1, sd=2) % D(sd=2) % F(sf=1), ADF_init)

    #Define the reactions the molecules in the model can undergo, with their
associated rates
    #Reaction 1: A + F | AF
    Rule('A_binds_F', A(sf=None, sd=None) + F(sf=None) | A(sf=1, sd=None) %
F(sf=1), kHybF, kHybR)
    #Reaction 2: A + D | AD
    Rule('A_binds_D', A(sf=None, sd=None) + D(sd=None) | A(sf=None, sd=2) %
D(sd=2), kAptF, kAptR)
    #Reaction 3: AF + D | ADF
    Rule('AF_binds_D', A(sf=1, sd=None) % F(sf=1) + D(sd=None) | A(sf=1, sd=2)
% D(sd=2) % F(sf=1), kAptIF_F, kAptIF_R)
    #Reaction 4: AD + F | ADF (The reverse of ADF | AD + F)
    Rule('AD_binds_F', A(sf=None, sd=2) % D(sd=2) + F(sf=None) | A(sf=1, sd=2)
% D(sd=2) % F(sf=1), kHybIF_F, kHybIF_R)


    #Define all the molecular species you want to be able to call, return, and
plot after simulating
    Observable('AF', A(sf=1, sd=None) % F(sf=1))
    Observable('F_unhyb', F(sf=None))
    Observable('D_unb', D(sd=None))
    Observable('A_unb', A(sf=None, sd=None))
    Observable('AD', A(sf=None, sd=2) % D(sd=2))
    Observable('ADF', A(sf=1, sd=2) % D(sd=2) % F(sf=1))

    #Return the model
    return a
```

```python
#Now, explain the reaction rate parameters
#We don't know any of the reaction rate constants.
#However, we have experimentally estimated that
#Kd_Hyb = kHybR / kHybF = 10 nM, and
#Kd_Apt = kHybR / kHybF = 30 μM

#The rate constants for the binding/dissociation reactions of the ADF intermediate are unknown.

#However, we can bet that, for an unstable intermediate, the forward (binding) reaction
#Will be equal to or lower than for the binding reaction of two species without the third.

#That is, kAptF ≥ kAptF*, for A + D --kAptF--> AD  and  AF + D --kAptF*--> ADF
#Likewise, kHybF ≥ kHybF*, for A + F --kHybF--> AF  and  AD + F --kHybF*--> ADF

#Similarly, we can bet that, for an unstable intermediate, the reverse (dissociation) reaction
#Will be equal to or higher than for the dissociation reaction of two species without the third.

#That is, kAptR ≤ kAptR*, for A + D <--kAptR-- AD  and  AF + D <--kAptR*-- ADF
#Likewise, kHybR ≤ kHybR*, for A + F <--kHybR-- AF  and  AD + F <--kHybR*-- ADF

#So, a reasonable way to constrain the initial parameter space is to say
#kHybR / kHybF = 10 nM
#kHybR / kHybF = 30 μM
#kAptF ≥ kAptF*
#kHybF ≥ kHybF*
#kAptR ≤ kAptR*
#kHybR ≤ kHybR*

#To start, let's say kHybF = 0.1, kHybR = 0.001, kAptF = 0.1, and kAptR = 3
#And kAptF* = 0.001, kAptR* = 10, kHybF* = 0.01, kHybR* = 0.01

#And for initial concentrations, assume 100 nM of aptamer-flare duplex.
#Also have 1 nM free aptamer and 1 nM free flare, just so beginning free flare concentrations ≠ 0.

```
```python
#All right, I've generated all the models. Now I want to write the method to auto-generate the simulation lists
#Inputs for genSimDF: a Model list mList, a TimeSpan t, and a list of observables to retrieve
```

```
def genSimDF(mList, t, variedParameter, observables):
    m_df = pd.DataFrame()
    mList_yOut = list()
    m_df['Time (s)'] = t

    for n in range(0, len(mList)):
        simRes     =     ScipyOdeSimulator(mList[n],     tspan     =     t,
compiler='python').run()
        mList_yOut.append(simRes.all)
        for i in range(0, len(observables)):

            m_df['[' + observables[i] + '], ' + variedParameter + ' = ' +
str(mList[n].parameters[variedParameter].value)]                                =
mList_yOut[n][observables[i]]

    paramsList = ['']*t.size
    rulesList = ['']*t.size
    m_params = mList[0].parameters[:]
    m_rules = mList[0].rules[:]
    for n in range(0,len(m_params)):
        paramsList[n] = m_params[n]

    for n in range(0, len(m_rules)):
        rulesList[n] = m_rules[n]

    m_df['Parameters'] = paramsList
    m_df['Rules'] = rulesList
    return m_df

```
```python
#Cool, so I've got DFs of all the simulations I might want.
#Now I want to generate fold-change-in-fluorescence dataframes.
def convertToFoldChange(df100D, dfNoD):
    numCols = df100D.shape[1] - 2
    FCdf = df100D.copy()

    for i in range(1, numCols):
        FCdf.iloc[:,i] = df100D.iloc[:,i] / dfNoD.iloc[:,i]

    return FCdf
```
```python
#Now generate plotting functions
#Something weird is happening; need to plot all the species to figure out what's
going on
def plotModelSims(modelDF, variedParam, paramList, yLabel, timespan):
    numCols = modelDF.shape[1] - 2
```

```
    colorDecimal = 1.0 / numCols
    pl.ion()
    for n in range(1,numCols):
        pl.plot(modelDF['Time (s)'][0:timespan],
                modelDF.iloc[0:timespan,n],
                label = (variedParam + ' = ' + str(paramList[n-1])),
                color=[colorDecimal*n,0.3,0.3])
    pl.legend()
    #Title the plot either 'Induced Fit' or 'Conformational Selection'
    if(modelDF.iloc[9,numCols] == ''):
        pl.title('Conformational Selection')
    else:
        pl.title('Induced Fit')

    pl.xlabel('Time (s)')
    pl.ylabel(yLabel)
    pl.figure()
```

```python
#Now, let's run plotting scripts to show the effects of varying all the different parameters
#First, generate lists of parameter values
#Span 5 orders of magnitude, by factors of 10
#use np.logspace(exponent1, exponent2, numPoints, base)

logBase10of3 = np.log(3)/np.log(10)
logBase10of5 = np.log(5)/np.log(10)

k_HybFlist = np.logspace(-4, 0, num=5, base=10)  #0.0001 - 1
k_HybRlist = np.logspace(-6, -2, num=5, base=10) #0.000001 - 0.01
k_AptFlist = np.logspace(-6, -1, num=6, base=10) #0.000001 - 0.1
k_AptRlist = np.logspace(logBase10of3 - 5, logBase10of3, num=6, base=10) #0.00003 - 3
k_AptIF_Flist = np.logspace(logBase10of5-7, logBase10of5-3, num=5, base=10) #0.0000005 - 0.005
k_AptIF_Rlist = np.logspace(-5, -1, num=5, base=10)                    #0.00001 - 0.1
k_HybIF_Flist = np.logspace(logBase10of5-5, logBase10of5-1, num=5, base=10) #0.00005 - 0.5
k_HybIF_Rlist = np.logspace(logBase10of5-6, logBase10of5-2, num=5, base=10) #0.000005 - 0.05

AF_0list = np.logspace(-3, 1, num=5, base=10)      #0.001 - 10
D_0list = np.logspace(-2, 4, num=7, base=10)      #0.01 - 10000
A_unb_0list = np.logspace(logBase10of5-4, logBase10of5, num=5, base=10) #0.0001 - 1
F_unhyb_0list = np.logspace(-4, 0, num=5, base=10) #0.0001 - 1
AD_0list = np.logspace(-4, 0, num=5, base=10)      #0.0001 - 1
```

```
ADF_0list = np.logspace(-6, -2, num=5, base=10)        #0.000001 - 0.01
```
```python
#Now show how these methods are deployed in practice.
#generate a list of numbers, varying parameter(s) of interest
#Possible reaction parameters to vary:
    #Both models: k_HybF, k_HybR, k_AptF, k_AptR
    #IF model only: k_HybIF_F, k_HybIF_R, k_AptIF_F, k_AptIF_R
#Possible initial concentrations to vary:
    #Both models: D_0, F_unhyb_0, A_unb_0, AF_0, AD_0
    #IF model only: ADF_0

D_0list = np.logspace(-2, 4, num=7, base=10)      #0.01 - 10000
k_AptRlist = np.logspace(logBase10of3 - 5, logBase10of3, num=6, base=10) #0.00003 - 3
#Next, generate a list of models, with each one varying by the parameters in the list just generated
#Show how to vary [D], and how to vary another parameter, while outputting fold change in fluorescence
CS_test_Dlist = [genAFCSmodel(D_0 = D_0list[n]) for n in range(0,len(D_0list))]
CS_test_kAptRlist_noD = [genAFCSmodel(k_AptR = k_AptRlist[n], D_0 = 0) for n in range(0,len(k_AptRlist))]
CS_test_kAptRlist_100D = [genAFCSmodel(k_AptR = k_AptRlist[n], D_0 = 100) for n in range(0,len(k_AptRlist))]
#Next, set some timespan parameters, and set a list of molecular species to observe (observables) during simulations
timeSpan = pl.linspace(0, 4000, num = 400)
allSpeciesIF = ['F_unhyb', 'AF', 'AD', 'ADF', 'A_unb'] #All the species observable in IF simulations
allSpeciesCS = ['F_unhyb', 'AF', 'AD', 'A_unb']           #All the species observable in CS simulations
FFonly = ['F_unhyb']   #Only observing [Free Flare]

#Now, generate a Pandas DataFrame containing Time (s), and the concentrations of all selected observables
#over the simulated time frame.
#The first column in the DataFrame is Time(s), and the last two columns list the starting parameters
#of the simulation, and the reactions/rules of the model being simulated
#Note that if varying the initial concentrations of molecular species, here are the variedParameter options:
    #Both models: D_init, F_unhyb_init, A_unb_init, AF_init, AD_init
    #IF model only: ADF_init
CS_test_D_df_FFonly = genSimDF(mList=CS_test_Dlist, t=timeSpan, variedParameter='D_init', observables=FFonly)
CS_test_kAptR_noD_df_FFonly = genSimDF(mList=CS_test_kAptRlist_noD, t=timeSpan, variedParameter='kAptR', observables=FFonly)
CS_test_kAptR_100D_df_FFonly = genSimDF(mList=CS_test_kAptRlist_100D, t=timeSpan, variedParameter='kAptR', observables=FFonly)
```

```
#Now, demonstrate how to convert to fold change in fluorescence for any parameter except D_0
CS_test_kAptR_100D_df_FoldChangeFF = convertToFoldChange(CS_test_kAptR_100D_df_FFonly, CS_test_kAptR_noD_df_FFonly)

#Now demonstrate how to convert to fold change in fluorescence for varying D_0
numCols = CS_test_D_df_FFonly.shape[1] - 2
CS_test_D_df_FoldChangeFF = CS_test_D_df_FFonly.copy()
for i in range(1, numCols):
    CS_test_D_df_FoldChangeFF.iloc[:,i] = CS_test_D_df_FFonly.iloc[:,i] / CS_test_D_df_FFonly.iloc[:,1]


#Now, demonstrate how to plot these models
plotModelSims(CS_test_D_df_FFonly, 'D_0', D_0list, yLabel = '[Free Flare], uM', timespan=4000)
plotModelSims(CS_test_D_df_FoldChangeFF, 'D_0', D_0list, yLabel = 'Fold Change in [Free Flare]', timespan=4000)
plotModelSims(CS_test_kAptR_noD_df_FFonly, 'No D, kAptR', D_0list, yLabel = '[Free Flare], uM', timespan=4000)
plotModelSims(CS_test_kAptR_100D_df_FFonly, '1000 uM D, kAptR', D_0list, yLabel = '[Free Flare], uM', timespan=4000)
plotModelSims(CS_test_kAptR_100D_df_FoldChangeFF, '1000 uM D, kAptR', D_0list, yLabel = 'Fold Change in [Free Flare]', timespan=4000)


#Now, demonstrate how to export the results of these models to an excel spreadsheet.
writer = pd.ExcelWriter('test_CSmodelSimAndExport.xlsx')

CS_test_D_df_FFonly.to_excel(writer,'Vary D_0, F only')
CS_test_D_df_FoldChangeFF.to_excel(writer,'Vary D_0, Fold Change F')
CS_test_kAptR_noD_df_FFonly.to_excel(writer,'Vary kAptR, 0 uM D, F only')
CS_test_kAptR_100D_df_FFonly.to_excel(writer,'Vary kAptR, 1000 uM D, F only')
CS_test_kAptR_100D_df_FoldChangeFF.to_excel(writer,'V kAptR, 1000 uM D, FoldChnge F')

writer.save()
```
```python
#And show how to check them for accuracy and how to export their structure to a PDF file
#First, export the generated models to flat .py files ('pysb_flat')
    #flat .py files just reproduce the model, don't contain options for simulating the models.
    #By contrast, exporting to .py returns a more sophisticated model that doesn't play well with render_reactions
    #But enables rapid and flexible simulation.
```

```
#.py files didn't work. maybe try .sbml files?
CS = genAFCSmodel()
IF = genAFIFmodel()

CS_pyFlat = export(CS, 'pysb_flat')
with open('CSmodelFlat.py', 'w') as f:
    f.write(CS_pyFlat)

IF_pyFlat = export(IF, 'pysb_flat')
with open('IFmodelFlat.py', 'w') as f:
    f.write(IF_pyFlat)

#Now, use PySB's render_reactions function to export these .py files to a .dot
file
!python       -m       pysb.tools.render_reactions       CSmodelFlat.py       >
jupyterPlots/CSmodel.dot
!python       -m       pysb.tools.render_reactions       IFmodelFlat.py       >
jupyterPlots/IFmodel.dot

#And now export those .dot files of the reaction networks to PDFs
!dot jupyterPlots/CSmodel.dot -T pdf -O
!dot jupyterPlots/IFmodel.dot -T pdf -O

#Excellent, this worked well

```
```python
#Now, how do I generalize the previous code into a method I can call to make
things easier?
#Excellent. Now I can export the dataframe to a csv file.
#NOTE that at the end of writing all these simulations to individual csv files,
I will also export them all to
#a single excel file, like so:
#writer = pd.ExcelWriter('output.xlsx')
#>>> df1.to_excel(writer,'Sheet1')
#>>> df2.to_excel(writer,'Sheet2')
#>>> writer.save()

#Now, to abstract all the work I've done before, these are the functions I'd
like to write:
#1. genSimDF: Given an data frame of models varying by 1 parameter, generate a
dataframe of simulations with parameters and rules as metadata
#2. plotSims: Given an array of simulations, plot them all, labeled by the
changing parameter, and varying in
    #shaded color intensity
#3. ExportSimDFtoCSV--this one I can probably just use to_csv for
#4. ExportSimDFstoExcel--this one I can probably also just use to_excel for
```

#5. Also want a way to show fraction of free flare (FracFF), and Fold Change in Free Flare (FoldChangeFF)

```
```python
#0. Varying the concentration of DHEA-S
CS_Dlist = [genAFCSmodel(D_0 = D_0list[n]) for n in range(0,7)]
IF_Dlist = [genAFIFmodel(D_0 = D_0list[n]) for n in range(0,7)]
```

```python
#Now, when varying other parameters, generate model lists for 100 μM DHEA-S and for 0 μM DHEA-S
#Now, what models do we want to make?
#1. Models varying kHybF
CS_kHybFnoD_list = [genAFCSmodel(k_HybF = k_HybFlist[n], D_0 = 0) for n in range(0,len(k_HybFlist))]
CS_kHybF100D_list = [genAFCSmodel(k_HybF = k_HybFlist[n], D_0 = 100) for n in range(0,len(k_HybFlist))]

IF_kHybFnoD_list = [genAFIFmodel(k_HybF = k_HybFlist[n], D_0 = 0) for n in range(0,len(k_HybFlist))]
IF_kHybF100D_list = [genAFIFmodel(k_HybF = k_HybFlist[n], D_0 = 100) for n in range(0,len(k_HybFlist))]
```

```python
#2. Model varying kHybR
CS_kHybRnoD_list = [genAFCSmodel(k_HybR = k_HybRlist[n], D_0 = 0) for n in range(0,len(k_HybRlist))]
CS_kHybR100D_list = [genAFCSmodel(k_HybR = k_HybRlist[n], D_0 = 100) for n in range(0,len(k_HybRlist))]

IF_kHybRnoD_list = [genAFIFmodel(k_HybR = k_HybRlist[n], D_0 = 0) for n in range(0,len(k_HybRlist))]
IF_kHybR100D_list = [genAFIFmodel(k_HybR = k_HybRlist[n], D_0 = 100) for n in range(0,len(k_HybRlist))]
#IF_kHybR100D_list[4].parameters

```

```python
#3. Model varying kAptF
CS_kAptFnoD_list = [genAFCSmodel(k_AptF = k_AptFlist[n], D_0 = 0) for n in range(0,len(k_AptFlist))]
CS_kAptF100D_list = [genAFCSmodel(k_AptF = k_AptFlist[n], D_0 = 100) for n in range(0,len(k_AptFlist))]
```
```

```
IF_kAptFnoD_list = [genAFIFmodel(k_AptF = k_AptFlist[n], D_0 = 0) for n in
range(0,len(k_AptFlist))]
IF_kAptF100D_list = [genAFIFmodel(k_AptF = k_AptFlist[n], D_0 = 100) for n in
range(0,len(k_AptFlist))]
#IF_kAptF100D_list[4].parameters
```

```python
#4. Model varying kAptR
CS_kAptRnoD_list = [genAFCSmodel(k_AptR = k_AptRlist[n], D_0 = 0) for n in
range(0,len(k_AptRlist))]
CS_kAptR100D_list = [genAFCSmodel(k_AptR = k_AptRlist[n], D_0 = 100) for n in
range(0,len(k_AptRlist))]

IF_kAptRnoD_list = [genAFIFmodel(k_AptR = k_AptRlist[n], D_0 = 0) for n in
range(0,len(k_AptRlist))]
IF_kAptR100D_list = [genAFIFmodel(k_AptR = k_AptRlist[n], D_0 = 100) for n in
range(0,len(k_AptRlist))]
#IF_kAptR100D_list[4].parameters
```

```python
#5. Model varying kAptR+F while keeping KdApt constant
CS_kApt_KdAptConst_noD_list = [genAFCSmodel(k_AptR = k_AptRlist[n],
                                            k_AptF = k_AptFlist[n],
                                            D_0    =    0)    for    n    in
range(0,len(k_AptRlist))]
CS_kApt_KdAptConst_100D_list = [genAFCSmodel(k_AptR = k_AptRlist[n],
                                             k_AptF = k_AptFlist[n],
                                             D_0    =    100)    for    n    in
range(0,len(k_AptRlist))]

IF_kApt_KdAptConst_noD_list = [genAFIFmodel(k_AptR = k_AptRlist[n],
                                            k_AptF = k_AptFlist[n],
                                            D_0    =    0)    for    n    in
range(0,len(k_AptRlist))]
IF_kApt_KdAptConst_100D_list = [genAFIFmodel(k_AptR = k_AptRlist[n],
                                             k_AptF = k_AptFlist[n],
                                             D_0    =    100)    for    n    in
range(0,len(k_AptRlist))]
#IF_kApt_KdAptConst_100D_list[4].parameters
```

```python
#6. Model varying kHybR while keeping KdHyb constant
CS_kHyb_KdHybConst_noD_list = [genAFCSmodel(k_HybR = k_HybRlist[n],
```

```
                                                k_HybF = k_HybFlist[n],
                                                D_0    =    0)    for    n    in
range(0,len(k_HybRlist))]
CS_kHyb_KdHybConst_100D_list = [genAFCSmodel(k_HybR = k_HybRlist[n],
                                                 k_HybF = k_HybFlist[n],
                                                 D_0    =    100)    for    n    in
range(0,len(k_HybRlist))]

IF_kHyb_KdHybConst_noD_list = [genAFIFmodel(k_HybR = k_HybRlist[n],
                                                k_HybF = k_HybFlist[n],
                                                D_0    =    0)    for    n    in
range(0,len(k_HybRlist))]
IF_kHyb_KdHybConst_100D_list = [genAFIFmodel(k_HybR = k_HybRlist[n],
                                                 k_HybF = k_HybFlist[n],
                                                 D_0    =    100)    for    n    in
range(0,len(k_HybRlist))]
#IF_kHyb_KdHybConst_100D_list[4].parameters
```

```python
#7. Model varying kHybF*
IF_kHybIF_F_noD_list = [genAFIFmodel(k_HybIF_F = k_HybIF_Flist[n], D_0 = 0) for
n in range(0,len(k_HybIF_Flist))]
IF_kHybIF_F_100D_list = [genAFIFmodel(k_HybIF_F = k_HybIF_Flist[n], D_0 = 100)
for n in range(0,len(k_HybIF_Flist))]
#IF_kHybIF_F_100D_list[0].parameters
```

```python
#8. Model varying kHybR*
IF_kHybIF_R_noD_list = [genAFIFmodel(k_HybIF_R = k_HybIF_Rlist[n], D_0 = 0) for
n in range(0,len(k_HybIF_Rlist))]
IF_kHybIF_R_100D_list = [genAFIFmodel(k_HybIF_R = k_HybIF_Rlist[n], D_0 = 100)
for n in range(0,len(k_HybIF_Rlist))]
#IF_kHybIF_R_100D_list[4].parameters
```

```python
#9. Model varying kAptF*
IF_kAptIF_F_noD_list = [genAFIFmodel(k_AptIF_F = k_AptIF_Flist[n], D_0 = 0) for
n in range(0,len(k_AptIF_Flist))]
IF_kAptIF_F_100D_list = [genAFIFmodel(k_AptIF_F = k_AptIF_Flist[n], D_0 = 100)
for n in range(0,len(k_AptIF_Flist))]
#IF_kAptIF_F_100D_list[4].parameters
```

```python
#10. Model varying kAptR*
IF_kAptIF_R_noD_list = [genAFIFmodel(k_AptIF_R = k_AptIF_Rlist[n], D_0 = 0) for n in range(0,len(k_AptIF_Rlist))]
IF_kAptIF_R_100D_list = [genAFIFmodel(k_AptIF_R = k_AptIF_Rlist[n], D_0 = 100) for n in range(0,len(k_AptIF_Rlist))]


```

```python
#Now, generate data frames from the model lists I created above
#For DHEA-S concentration ranges, make all species DFs and a DF with just Free Flare (FF)
#Actually, just do that for all of them; why not?
tspan = pl.linspace(0, 4000, num = 400)
allSpeciesIF = ['F_unhyb', 'AF', 'AD', 'ADF', 'A_unb']
allSpeciesCS = ['F_unhyb', 'AF', 'AD', 'A_unb']
FFonly = ['F_unhyb']
```

```python
#0. Models varying [DHEA-S]
CS_D_df_allSpecies = genSimDF(mList=CS_Dlist, t=tspan, variedParameter='D_init', observables=allSpeciesCS)
CS_D_df_FFonly = genSimDF(mList=CS_Dlist, t=tspan, variedParameter='D_init', observables=FFonly)

IF_D_df_allSpecies = genSimDF(mList=IF_Dlist, t=tspan, variedParameter='D_init', observables=allSpeciesIF)
IF_D_df_FFonly = genSimDF(mList=IF_Dlist, t=tspan, variedParameter='D_init', observables=FFonly)
#IF_D_df_allSpecies
```

```python
#1. Models varying kHybF
CS_kHybFnoD_df_allSpecies = genSimDF(mList=CS_kHybFnoD_list, t=tspan, variedParameter='kHybF', observables=allSpeciesCS)
CS_kHybF100D_df_allSpecies = genSimDF(mList=CS_kHybF100D_list, t=tspan, variedParameter='kHybF', observables=allSpeciesCS)

CS_kHybFnoD_df_FFonly = genSimDF(mList=CS_kHybFnoD_list, t=tspan, variedParameter='kHybF', observables=FFonly)
CS_kHybF100D_df_FFonly = genSimDF(mList=CS_kHybF100D_list, t=tspan, variedParameter='kHybF', observables=FFonly)
```

```
IF_kHybFnoD_df_allSpecies    =    genSimDF(mList=IF_kHybFnoD_list,    t=tspan,
variedParameter='kHybF', observables=allSpeciesCS)
IF_kHybF100D_df_allSpecies   =    genSimDF(mList=IF_kHybF100D_list,   t=tspan,
variedParameter='kHybF', observables=allSpeciesCS)

IF_kHybFnoD_df_FFonly     =     genSimDF(mList=IF_kHybFnoD_list,     t=tspan,
variedParameter='kHybF', observables=FFonly)
IF_kHybF100D_df_FFonly    =     genSimDF(mList=IF_kHybF100D_list,    t=tspan,
variedParameter='kHybF', observables=FFonly)

```

```python
#2. Models varying kHybR
CS_kHybRnoD_df_allSpecies    =    genSimDF(mList=CS_kHybRnoD_list,    t=tspan,
variedParameter='kHybR', observables=allSpeciesCS)
CS_kHybR100D_df_allSpecies   =    genSimDF(mList=CS_kHybR100D_list,   t=tspan,
variedParameter='kHybR', observables=allSpeciesCS)

CS_kHybRnoD_df_FFonly     =     genSimDF(mList=CS_kHybRnoD_list,     t=tspan,
variedParameter='kHybR', observables=FFonly)
CS_kHybR100D_df_FFonly    =     genSimDF(mList=CS_kHybR100D_list,    t=tspan,
variedParameter='kHybR', observables=FFonly)

IF_kHybRnoD_df_allSpecies    =    genSimDF(mList=IF_kHybRnoD_list,    t=tspan,
variedParameter='kHybR', observables=allSpeciesCS)
IF_kHybR100D_df_allSpecies   =    genSimDF(mList=IF_kHybR100D_list,   t=tspan,
variedParameter='kHybR', observables=allSpeciesCS)

IF_kHybRnoD_df_FFonly     =     genSimDF(mList=IF_kHybRnoD_list,     t=tspan,
variedParameter='kHybR', observables=FFonly)
IF_kHybR100D_df_FFonly    =     genSimDF(mList=IF_kHybR100D_list,    t=tspan,
variedParameter='kHybR', observables=FFonly)

```

```python
#3. Models varying kAptF
CS_kAptFnoD_df_allSpecies    =    genSimDF(mList=CS_kAptFnoD_list,    t=tspan,
variedParameter='kAptF', observables=allSpeciesCS)
CS_kAptF100D_df_allSpecies   =    genSimDF(mList=CS_kAptF100D_list,   t=tspan,
variedParameter='kAptF', observables=allSpeciesCS)

CS_kAptFnoD_df_FFonly     =     genSimDF(mList=CS_kAptFnoD_list,     t=tspan,
variedParameter='kAptF', observables=FFonly)
CS_kAptF100D_df_FFonly    =     genSimDF(mList=CS_kAptF100D_list,    t=tspan,
variedParameter='kAptF', observables=FFonly)

IF_kAptFnoD_df_allSpecies    =    genSimDF(mList=IF_kAptFnoD_list,    t=tspan,
variedParameter='kAptF', observables=allSpeciesCS)
```

```
IF_kAptF100D_df_allSpecies    =    genSimDF(mList=IF_kAptF100D_list,    t=tspan,
variedParameter='kAptF', observables=allSpeciesCS)

IF_kAptFnoD_df_FFonly     =     genSimDF(mList=IF_kAptFnoD_list,     t=tspan,
variedParameter='kAptF', observables=FFonly)
IF_kAptF100D_df_FFonly    =     genSimDF(mList=IF_kAptF100D_list,    t=tspan,
variedParameter='kAptF', observables=FFonly)

```

```python
#4. Models varying kAptR
CS_kAptRnoD_df_allSpecies    =    genSimDF(mList=CS_kAptRnoD_list,    t=tspan,
variedParameter='kAptR', observables=allSpeciesCS)
CS_kAptR100D_df_allSpecies   =    genSimDF(mList=CS_kAptR100D_list,   t=tspan,
variedParameter='kAptR', observables=allSpeciesCS)

CS_kAptRnoD_df_FFonly     =     genSimDF(mList=CS_kAptRnoD_list,     t=tspan,
variedParameter='kAptR', observables=FFonly)
CS_kAptR100D_df_FFonly    =     genSimDF(mList=CS_kAptR100D_list,    t=tspan,
variedParameter='kAptR', observables=FFonly)

IF_kAptRnoD_df_allSpecies    =    genSimDF(mList=IF_kAptRnoD_list,    t=tspan,
variedParameter='kAptR', observables=allSpeciesCS)
IF_kAptR100D_df_allSpecies   =    genSimDF(mList=IF_kAptR100D_list,   t=tspan,
variedParameter='kAptR', observables=allSpeciesCS)

IF_kAptRnoD_df_FFonly     =     genSimDF(mList=IF_kAptRnoD_list,     t=tspan,
variedParameter='kAptR', observables=FFonly)
IF_kAptR100D_df_FFonly    =     genSimDF(mList=IF_kAptR100D_list,    t=tspan,
variedParameter='kAptR', observables=FFonly)

```

```python
#5. Models varying kHybR while keeping KdHyb constant
CS_kHyb_KdHybConst_noD_df_allSpecies                                    =
genSimDF(mList=CS_kHyb_KdHybConst_noD_list, t=tspan, variedParameter='kHybR',
observables=allSpeciesCS)
CS_kHyb_KdHybConst_100D_df_allSpecies                                   =
genSimDF(mList=CS_kHyb_KdHybConst_100D_list, t=tspan, variedParameter='kHybR',
observables=allSpeciesCS)

CS_kHyb_KdHybConst_noD_df_FFonly = genSimDF(mList=CS_kHyb_KdHybConst_noD_list,
t=tspan, variedParameter='kHybR', observables=FFonly)
CS_kHyb_KdHybConst_100D_df_FFonly                                       =
genSimDF(mList=CS_kHyb_KdHybConst_100D_list, t=tspan, variedParameter='kHybR',
observables=FFonly)
```

```
IF_kHyb_KdHybConst_noD_df_allSpecies                                    =
genSimDF(mList=IF_kHyb_KdHybConst_noD_list, t=tspan, variedParameter='kHybR',
observables=allSpeciesCS)
IF_kHyb_KdHybConst_100D_df_allSpecies                                   =
genSimDF(mList=IF_kHyb_KdHybConst_100D_list, t=tspan, variedParameter='kHybR',
observables=allSpeciesCS)

IF_kHyb_KdHybConst_noD_df_FFonly = genSimDF(mList=IF_kHyb_KdHybConst_noD_list,
t=tspan, variedParameter='kHybR', observables=FFonly)
IF_kHyb_KdHybConst_100D_df_FFonly                                       =
genSimDF(mList=IF_kHyb_KdHybConst_100D_list, t=tspan, variedParameter='kHybR',
observables=FFonly)
```

```python
#6. Models varying kAptR while keeping KdApt constant
CS_kApt_KdAptConst_noD_df_allSpecies                                    =
genSimDF(mList=CS_kApt_KdAptConst_noD_list, t=tspan, variedParameter='kAptR',
observables=allSpeciesCS)
CS_kApt_KdAptConst_100D_df_allSpecies                                   =
genSimDF(mList=CS_kApt_KdAptConst_100D_list, t=tspan, variedParameter='kAptR',
observables=allSpeciesCS)

CS_kApt_KdAptConst_noD_df_FFonly = genSimDF(mList=CS_kApt_KdAptConst_noD_list,
t=tspan, variedParameter='kAptR', observables=FFonly)
CS_kApt_KdAptConst_100D_df_FFonly                                       =
genSimDF(mList=CS_kApt_KdAptConst_100D_list, t=tspan, variedParameter='kAptR',
observables=FFonly)

IF_kApt_KdAptConst_noD_df_allSpecies                                    =
genSimDF(mList=IF_kApt_KdAptConst_noD_list, t=tspan, variedParameter='kAptR',
observables=allSpeciesCS)
IF_kApt_KdAptConst_100D_df_allSpecies                                   =
genSimDF(mList=IF_kApt_KdAptConst_100D_list, t=tspan, variedParameter='kAptR',
observables=allSpeciesCS)

IF_kApt_KdAptConst_noD_df_FFonly = genSimDF(mList=IF_kApt_KdAptConst_noD_list,
t=tspan, variedParameter='kAptR', observables=FFonly)
IF_kApt_KdAptConst_100D_df_FFonly                                       =
genSimDF(mList=IF_kApt_KdAptConst_100D_list, t=tspan, variedParameter='kAptR',
observables=FFonly)
```

```python
#7. Model varying kHybF*
IF_kHybIF_F_noD_df_allSpecies = genSimDF(mList=IF_kHybIF_F_noD_list, t=tspan,
variedParameter='kHybIF_F', observables=allSpeciesCS)
IF_kHybIF_F_100D_df_allSpecies     =     genSimDF(mList=IF_kHybIF_F_100D_list,
t=tspan, variedParameter='kHybIF_F', observables=allSpeciesCS)
```

```
IF_kHybIF_F_noD_df_FFonly = genSimDF(mList=IF_kHybIF_F_noD_list, t=tspan, variedParameter='kHybIF_F', observables=FFonly)
IF_kHybIF_F_100D_df_FFonly = genSimDF(mList=IF_kHybIF_F_100D_list, t=tspan, variedParameter='kHybIF_F', observables=FFonly)
```

```python
#8. Model varying kHybR*
IF_kHybIF_R_noD_df_allSpecies = genSimDF(mList=IF_kHybIF_R_noD_list, t=tspan, variedParameter='kHybIF_R', observables=allSpeciesCS)
IF_kHybIF_R_100D_df_allSpecies = genSimDF(mList=IF_kHybIF_R_100D_list, t=tspan, variedParameter='kHybIF_R', observables=allSpeciesCS)

IF_kHybIF_R_noD_df_FFonly = genSimDF(mList=IF_kHybIF_R_noD_list, t=tspan, variedParameter='kHybIF_R', observables=FFonly)
IF_kHybIF_R_100D_df_FFonly = genSimDF(mList=IF_kHybIF_R_100D_list, t=tspan, variedParameter='kHybIF_R', observables=FFonly)
```

```python
#9. Model varying kAptF*
IF_kAptIF_F_noD_df_allSpecies = genSimDF(mList=IF_kAptIF_F_noD_list, t=tspan, variedParameter='kAptIF_F', observables=allSpeciesCS)
IF_kAptIF_F_100D_df_allSpecies = genSimDF(mList=IF_kAptIF_F_100D_list, t=tspan, variedParameter='kAptIF_F', observables=allSpeciesCS)

IF_kAptIF_F_noD_df_FFonly = genSimDF(mList=IF_kAptIF_F_noD_list, t=tspan, variedParameter='kAptIF_F', observables=FFonly)
IF_kAptIF_F_100D_df_FFonly = genSimDF(mList=IF_kAptIF_F_100D_list, t=tspan, variedParameter='kAptIF_F', observables=FFonly)
```

```python
#10. Model varying kAptR*
IF_kAptIF_R_noD_df_allSpecies = genSimDF(mList=IF_kAptIF_R_noD_list, t=tspan, variedParameter='kAptIF_R', observables=allSpeciesCS)
IF_kAptIF_R_100D_df_allSpecies = genSimDF(mList=IF_kAptIF_R_100D_list, t=tspan, variedParameter='kAptIF_R', observables=allSpeciesCS)

IF_kAptIF_R_noD_df_FFonly = genSimDF(mList=IF_kAptIF_R_noD_list, t=tspan, variedParameter='kAptIF_R', observables=FFonly)
IF_kAptIF_R_100D_df_FFonly = genSimDF(mList=IF_kAptIF_R_100D_list, t=tspan, variedParameter='kAptIF_R', observables=FFonly)
```

```python
#Also make custon dataframes for the Dlist simulations to calculate fold change in fluorescence
numCols = CS_D_df_FFonly.shape[1] - 2
CS_D_df_FoldChange = CS_D_df_FFonly.copy()
```

```
IF_D_df_FoldChange = IF_D_df_FFonly.copy()
for i in range(1, numCols):
    CS_D_df_FoldChange.iloc[:,i]       =       CS_D_df_FFonly.iloc[:,i]       /
CS_D_df_FFonly.iloc[:,1]
    IF_D_df_FoldChange.iloc[:,i]       =       IF_D_df_FFonly.iloc[:,i]       /
IF_D_df_FFonly.iloc[:,1]

```

```python
#CS_D_df_FoldChange
#CS_D_df_FFonly
```

```python
#Now, generate all the foldChange dataframes
CS_kHybF_df_FoldChange      =      convertToFoldChange(CS_kHybF100D_df_FFonly,
CS_kHybFnoD_df_FFonly)
CS_kHybR_df_FoldChange      =      convertToFoldChange(CS_kHybR100D_df_FFonly,
CS_kHybRnoD_df_FFonly)

IF_kHybF_df_FoldChange      =      convertToFoldChange(IF_kHybF100D_df_FFonly,
IF_kHybFnoD_df_FFonly)
IF_kHybR_df_FoldChange      =      convertToFoldChange(IF_kHybR100D_df_FFonly,
IF_kHybRnoD_df_FFonly)

CS_kAptF_df_FoldChange      =      convertToFoldChange(CS_kAptF100D_df_FFonly,
CS_kAptFnoD_df_FFonly)
CS_kAptR_df_FoldChange      =      convertToFoldChange(CS_kAptR100D_df_FFonly,
CS_kAptRnoD_df_FFonly)

IF_kAptF_df_FoldChange      =      convertToFoldChange(IF_kAptF100D_df_FFonly,
IF_kAptFnoD_df_FFonly)
IF_kAptR_df_FoldChange      =      convertToFoldChange(IF_kAptR100D_df_FFonly,
IF_kAptRnoD_df_FFonly)

CS_kHyb_KdHybConst_df_FoldChange                                              =
convertToFoldChange(CS_kHyb_KdHybConst_100D_df_FFonly,
CS_kHyb_KdHybConst_noD_df_FFonly)
CS_kApt_KdAptConst_df_FoldChange                                              =
convertToFoldChange(CS_kApt_KdAptConst_100D_df_FFonly,
CS_kApt_KdAptConst_noD_df_FFonly)
IF_kHyb_KdHybConst_df_FoldChange                                              =
convertToFoldChange(IF_kHyb_KdHybConst_100D_df_FFonly,
IF_kHyb_KdHybConst_noD_df_FFonly)
IF_kApt_KdAptConst_df_FoldChange                                              =
convertToFoldChange(IF_kApt_KdAptConst_100D_df_FFonly,
IF_kApt_KdAptConst_noD_df_FFonly)
```

```
IF_kHybIF_F_df_FoldChange = convertToFoldChange(IF_kHybIF_F_100D_df_FFonly, IF_kHybIF_F_noD_df_FFonly)
IF_kHybIF_R_df_FoldChange = convertToFoldChange(IF_kHybIF_R_100D_df_FFonly, IF_kHybIF_R_noD_df_FFonly)

IF_kAptIF_F_df_FoldChange = convertToFoldChange(IF_kAptIF_F_100D_df_FFonly, IF_kAptIF_F_noD_df_FFonly)
IF_kAptIF_R_df_FoldChange = convertToFoldChange(IF_kAptIF_R_100D_df_FFonly, IF_kAptIF_R_noD_df_FFonly)
```

```python
#Plot all the varied parameters
plotModelSims(CS_D_df_FoldChange, 'D_0', D_0list, yLabel = 'Fold Change in [Free Flare]', timespan=4000)
plotModelSims(IF_D_df_FoldChange, 'D_0', D_0list, yLabel = 'Fold Change in [Free Flare]', timespan=4000)
plotModelSims(CS_kHybF_df_FoldChange, 'kHybF', k_HybFlist, yLabel = 'Fold Change in [Free Flare]', timespan=4000)
plotModelSims(IF_kHybF_df_FoldChange, 'kHybF', k_HybFlist, yLabel = 'Fold Change in [Free Flare]', timespan=4000)
plotModelSims(CS_kHybR_df_FoldChange, 'kHybR', k_HybRlist, yLabel = 'Fold Change in [Free Flare]', timespan=4000)
plotModelSims(IF_kHybR_df_FoldChange, 'kHybR', k_HybRlist, yLabel = 'Fold Change in [Free Flare]', timespan=4000)

plotModelSims(CS_kAptF_df_FoldChange, 'kAptF', k_AptFlist, yLabel = 'Fold Change in [Free Flare]', timespan=4000)
plotModelSims(CS_kAptR_df_FoldChange, 'kAptR', k_AptRlist, yLabel = 'Fold Change in [Free Flare]', timespan=4000)
plotModelSims(IF_kAptF_df_FoldChange, 'kAptF', k_AptFlist, yLabel = 'Fold Change in [Free Flare]', timespan=4000)
plotModelSims(IF_kAptR_df_FoldChange, 'kAptR', k_AptRlist, yLabel = 'Fold Change in [Free Flare]', timespan=4000)

plotModelSims(CS_kHyb_KdHybConst_df_FoldChange, 'KdHyb=10nM, kHybR', k_HybRlist, yLabel = 'Fold Change in [Free Flare]', timespan=4000)
plotModelSims(IF_kHyb_KdHybConst_df_FoldChange, 'KdHyb=10nM, kHybR', k_HybRlist, yLabel = 'Fold Change in [Free Flare]', timespan=4000)
plotModelSims(CS_kApt_KdAptConst_df_FoldChange, 'KdApt=30uM, kAptR', k_AptRlist, yLabel = 'Fold Change in [Free Flare]', timespan=4000)
plotModelSims(IF_kApt_KdAptConst_df_FoldChange, 'KdApt=30uM, kAptR', k_AptRlist, yLabel = 'Fold Change in [Free Flare]', timespan=4000)

plotModelSims(IF_kHybIF_F_df_FoldChange, 'kHybF*', k_HybIF_Flist, yLabel = 'Fold Change in [Free Flare]', timespan=4000)
plotModelSims(IF_kHybIF_R_df_FoldChange, 'kHybR*', k_HybIF_Rlist, yLabel = 'Fold Change in [Free Flare]', timespan=4000)
plotModelSims(IF_kAptIF_F_df_FoldChange, 'kAptF*', k_AptIF_Flist, yLabel = 'Fold Change in [Free Flare]', timespan=4000)
```

```
plotModelSims(IF_kAptIF_R_df_FoldChange, 'kAptR*', k_AptIF_Rlist, yLabel =
'Fold Change in [Free Flare]', timespan=4000)
```

```
```python
#Now let's export all this simulation data to excel sheets, and then send it to
Peter and Jorge
```

```python
#First, the Conformational Selection, All Molecular Species spreadsheet
writer = pd.ExcelWriter('20190205_CS_ModelOutputs_AllConcentrations.xlsx')
CS_D_df_allSpecies.to_excel(writer,'Vary D_0')
CS_kHybFnoD_df_allSpecies.to_excel(writer,'Vary kHybF (no D)')
CS_kHybF100D_df_allSpecies.to_excel(writer,'Vary kHybF (100uM D)')
CS_kHybRnoD_df_allSpecies.to_excel(writer,'Vary kHybR (no D)')
CS_kHybR100D_df_allSpecies.to_excel(writer,'Vary kHybR (100uM D)')
CS_kAptFnoD_df_allSpecies.to_excel(writer,'Vary kAptF (no D)')
CS_kAptF100D_df_allSpecies.to_excel(writer,'Vary kAptF (100uM D)')
CS_kAptRnoD_df_allSpecies.to_excel(writer,'Vary kAptR (no D)')
CS_kAptR100D_df_allSpecies.to_excel(writer,'Vary kAptR (100uM D)')
CS_kHyb_KdHybConst_noD_df_allSpecies.to_excel(writer,'KdHyb=10nM, kHyb (no
D)')
CS_kHyb_KdHybConst_100D_df_allSpecies.to_excel(writer,'KdHyb=10nM, kHyb (100uM
D)')
CS_kApt_KdAptConst_noD_df_allSpecies.to_excel(writer,'KdApt=10nM, kApt (no
D)')
CS_kApt_KdAptConst_100D_df_allSpecies.to_excel(writer,'KdApt=10nM, kApt (100uM
D)')
writer.save()
```

```python
#Next, the Induced Fit, All Molecular Species spreadsheet
#First, the Conformational Selection, All Molecular Species spreadsheet
writer = pd.ExcelWriter('20190205_IF_ModelOutputs_AllConcentrations.xlsx')
IF_D_df_allSpecies.to_excel(writer,'Vary D_0')
IF_kHybFnoD_df_allSpecies.to_excel(writer,'Vary kHybF (no D)')
IF_kHybF100D_df_allSpecies.to_excel(writer,'Vary kHybF (100uM D)')
IF_kHybRnoD_df_allSpecies.to_excel(writer,'Vary kHybR (no D)')
IF_kHybR100D_df_allSpecies.to_excel(writer,'Vary kHybR (100uM D)')
IF_kAptFnoD_df_allSpecies.to_excel(writer,'Vary kAptF (no D)')
IF_kAptF100D_df_allSpecies.to_excel(writer,'Vary kAptF (100uM D)')
IF_kAptRnoD_df_allSpecies.to_excel(writer,'Vary kAptR (no D)')
IF_kAptR100D_df_allSpecies.to_excel(writer,'Vary kAptR (100uM D)')
IF_kHyb_KdHybConst_noD_df_allSpecies.to_excel(writer,'KdHyb=10nM, kHyb (no
D)')
```
```

```
IF_kHyb_KdHybConst_100D_df_allSpecies.to_excel(writer,'KdHyb=10nM, kHyb (100uM
D)')
IF_kApt_KdAptConst_noD_df_allSpecies.to_excel(writer,'KdApt=10nM,   kApt   (no
D)')
IF_kApt_KdAptConst_100D_df_allSpecies.to_excel(writer,'KdApt=10nM, kApt (100uM
D)')

IF_kHybIF_F_noD_df_allSpecies.to_excel(writer,'kHybIF_F (no D)')
IF_kHybIF_R_100D_df_allSpecies.to_excel(writer,'kHybIF_R (100uM D)')
IF_kAptIF_F_noD_df_allSpecies.to_excel(writer,'kAptIF_F (no D)')
IF_kAptIF_R_100D_df_allSpecies.to_excel(writer,'kAptIF_R (100uM D)')
writer.save()
```

```python
#Now, the Conformational Selection, Only [Free Flare] spreadsheet
writer = pd.ExcelWriter('IF_ModelOutputs_FFonly.xlsx')
CS_D_df_FFonly.to_excel(writer,'Vary D_0')
CS_kHybFnoD_df_FFonly.to_excel(writer,'Vary kHybF (no D)')
CS_kHybF100D_df_FFonly.to_excel(writer,'Vary kHybF (100uM D)')
CS_kHybRnoD_df_FFonly.to_excel(writer,'Vary kHybR (no D)')
CS_kHybR100D_df_FFonly.to_excel(writer,'Vary kHybR (100uM D)')
CS_kAptFnoD_df_FFonly.to_excel(writer,'Vary kAptF (no D)')
CS_kAptF100D_df_FFonly.to_excel(writer,'Vary kAptF (100uM D)')
CS_kAptRnoD_df_FFonly.to_excel(writer,'Vary kAptR (no D)')
CS_kAptR100D_df_FFonly.to_excel(writer,'Vary kAptR (100uM D)')
CS_kHyb_KdHybConst_noD_df_FFonly.to_excel(writer,'KdHyb=10nM, kHyb (no D)')
CS_kHyb_KdHybConst_100D_df_FFonly.to_excel(writer,'KdHyb=10nM,   kHyb   (100uM
D)')
CS_kApt_KdAptConst_noD_df_FFonly.to_excel(writer,'KdApt=10nM, kApt (no D)')
CS_kApt_KdAptConst_100D_df_FFonly.to_excel(writer,'KdApt=10nM,   kApt   (100uM
D)')

writer.save()
```

```python
#Now, the Induced Fit, Only [Free Flare] spreadsheet
writer = pd.ExcelWriter('20190205_IF_ModelOutputs_FFonly.xlsx')
IF_D_df_FFonly.to_excel(writer,'Vary D_0')
IF_kHybFnoD_df_FFonly.to_excel(writer,'Vary kHybF (no D)')
IF_kHybF100D_df_FFonly.to_excel(writer,'Vary kHybF (100uM D)')
IF_kHybRnoD_df_FFonly.to_excel(writer,'Vary kHybR (no D)')
IF_kHybR100D_df_FFonly.to_excel(writer,'Vary kHybR (100uM D)')
IF_kAptFnoD_df_FFonly.to_excel(writer,'Vary kAptF (no D)')
IF_kAptF100D_df_FFonly.to_excel(writer,'Vary kAptF (100uM D)')
IF_kAptRnoD_df_FFonly.to_excel(writer,'Vary kAptR (no D)')
IF_kAptR100D_df_FFonly.to_excel(writer,'Vary kAptR (100uM D)')
```

```
IF_kHyb_KdHybConst_noD_df_FFonly.to_excel(writer,'KdHyb=10nM, kHyb (no D)')
IF_kHyb_KdHybConst_100D_df_FFonly.to_excel(writer,'KdHyb=10nM,   kHyb   (100uM
D)')
IF_kApt_KdAptConst_noD_df_FFonly.to_excel(writer,'KdApt=10nM, kApt (no D)')
IF_kApt_KdAptConst_100D_df_FFonly.to_excel(writer,'KdApt=10nM,   kApt   (100uM
D)')

IF_kHybIF_F_noD_df_FFonly.to_excel(writer,'kHybIF_F (no D)')
IF_kHybIF_R_100D_df_FFonly.to_excel(writer,'kHybIF_R (100uM D)')
IF_kAptIF_F_noD_df_FFonly.to_excel(writer,'kAptIF_F (no D)')
IF_kAptIF_R_100D_df_FFonly.to_excel(writer,'kAptIF_R (100uM D)')
writer.save()
```

```python
#Now, the Conformational Selection, Fold Change in [Free Flare] spreadsheet
writer = pd.ExcelWriter('20190205_CS_ModelOutputs_FoldChangeFF.xlsx')
CS_D_df_FoldChange.to_excel(writer,'Vary D_0')
CS_kHybF_df_FoldChange.to_excel(writer,'Vary kHybF')
CS_kHybR_df_FoldChange.to_excel(writer,'Vary kHybR')
CS_kAptF_df_FoldChange.to_excel(writer,'Vary kAptF')
CS_kAptR_df_FoldChange.to_excel(writer,'Vary kAptR')
CS_kHyb_KdHybConst_df_FoldChange.to_excel(writer,'KdHyb=10nM, Vary kHyb')
CS_kApt_KdAptConst_df_FoldChange.to_excel(writer,'KdApt=10nM, Vary kApt')
writer.save()
```

```python
#Now, the Conformational Selection, Fold Change in [Free Flare] spreadsheet
writer = pd.ExcelWriter('20190205_IF_ModelOutputs_FoldChangeFF.xlsx')
IF_D_df_FoldChange.to_excel(writer,'Vary D_0')
IF_kHybF_df_FoldChange.to_excel(writer,'Vary kHybF')
IF_kHybR_df_FoldChange.to_excel(writer,'Vary kHybR')
IF_kAptF_df_FoldChange.to_excel(writer,'Vary kAptF')
IF_kAptR_df_FoldChange.to_excel(writer,'Vary kAptR')
IF_kHyb_KdHybConst_df_FoldChange.to_excel(writer,'KdHyb=10nM, Vary kHyb')
IF_kApt_KdAptConst_df_FoldChange.to_excel(writer,'KdApt=10nM, Vary kApt')
IF_kHybIF_F_df_FoldChange.to_excel(writer,'Vary kHybIF_F')
IF_kHybIF_R_df_FoldChange.to_excel(writer,'Vary kHybIF_R')
IF_kAptIF_F_df_FoldChange.to_excel(writer,'Vary kAptIF_F')
IF_kAptIF_R_df_FoldChange.to_excel(writer,'Vary kAptIF_R')
writer.save()
```

```python
#Now, the Conformational Selection, Fold Change in [Free Flare] spreadsheet
writer = pd.ExcelWriter('20190205_IF+CS_ModelOutputs_FoldChangeFF.xlsx')
```

```
IF_D_df_FoldChange.to_excel(writer,'IF, Vary D_0')
CS_D_df_FoldChange.to_excel(writer,'CS, Vary D_0')
IF_kHybF_df_FoldChange.to_excel(writer,'IF, Vary kHybF')
CS_kHybF_df_FoldChange.to_excel(writer,'CS, Vary kHybF')
IF_kHybR_df_FoldChange.to_excel(writer,'IF, Vary kHybR')
CS_kHybR_df_FoldChange.to_excel(writer,'CS, Vary kHybR')
IF_kAptF_df_FoldChange.to_excel(writer,'IF, Vary kAptF')
CS_kAptF_df_FoldChange.to_excel(writer,'CS, Vary kAptF')
IF_kAptR_df_FoldChange.to_excel(writer,'IF, Vary kAptR')
CS_kAptR_df_FoldChange.to_excel(writer,'CS, Vary kAptR')
IF_kHyb_KdHybConst_df_FoldChange.to_excel(writer,'IF, KdHyb=10nM, Vary kHyb')
CS_kHyb_KdHybConst_df_FoldChange.to_excel(writer,'CS, KdHyb=10nM, Vary kHyb')
IF_kApt_KdAptConst_df_FoldChange.to_excel(writer,'IF, KdApt=10nM, Vary kApt')
CS_kApt_KdAptConst_df_FoldChange.to_excel(writer,'CS, KdApt=10nM, Vary kApt')

IF_kHybIF_F_df_FoldChange.to_excel(writer,'IF, Vary kHybIF_F')
IF_kHybIF_R_df_FoldChange.to_excel(writer,'IF, Vary kHybIF_R')
IF_kAptIF_F_df_FoldChange.to_excel(writer,'IF, Vary kAptIF_F')
IF_kAptIF_R_df_FoldChange.to_excel(writer,'IF, Vary kAptIF_R')
writer.save()
```

Code A3. Microarray data processing markdown file.

```
```python
#Refactored version of the 20170709 notebook to make it easier to follow.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time
```
```python
#Oh, wait. I'm forgetting to find all the improperly named ACEs.
#Now I want a way to generate the reverse complement of all the aptamer sequences. Do that.
def revComp(seq):
    seqLength = len(seq)
    #print seqLength
    revSeq = seq[::-1]
    #print 'revSeq = ' + revSeq
    #print 'len(revSeq) = ' + str(len(revSeq))
    revCompSeq = ''
    for i in range(0, seqLength):
        currentBase = revSeq[i]
        if(currentBase == 'A' or currentBase == 'a'):
            revCompSeq = revCompSeq + 'T'
        if(currentBase == 'T' or currentBase == 't'):
            revCompSeq = revCompSeq + 'A'
        if(currentBase == 'G' or currentBase == 'g'):
            revCompSeq = revCompSeq + 'C'
        if(currentBase == 'C' or currentBase == 'c'):
            revCompSeq = revCompSeq + 'G'
    return revCompSeq

def revComp_RNA(seq):
    seqLength = len(seq)
    #print seqLength
    revSeq = seq[::-1]
    #print 'revSeq = ' + revSeq
    #print 'len(revSeq) = ' + str(len(revSeq))
    revCompSeq = ''
    for i in range(0, seqLength):
        currentBase = revSeq[i]
        if(currentBase == 'A' or currentBase == 'a'):
            revCompSeq = revCompSeq + 'U'
        if(currentBase == 'U' or currentBase == 'U'):
            revCompSeq = revCompSeq + 'A'
        if(currentBase == 'G' or currentBase == 'g'):
            revCompSeq = revCompSeq + 'C'
        if(currentBase == 'C' or currentBase == 'c'):
```
```

```
            revCompSeq = revCompSeq + 'G'
    return revCompSeq

#Let's just make a quick function to make it quicker and easier to return the
sequence of an aptamer
#From the aptamer array
def returnAptSeq(aptName, aptArray):
    aptSeq          =          aptArray[aptArray['uniqueAptNames']          ==
aptName].iloc[0]['uniqueAptSequences']
    return aptSeq

#Great, now make a quick method to automatically generate the number of ACEs,
given an aptamer length, a minimum
#ACE length, and a maximum ACE length
def numACEs(aptLen, minACElen, maxACElen):
    ACEcounter = 0
    for i in range(minACElen, maxACElen+1):
        ACEcounter = ACEcounter + aptLen - i + 1
    return ACEcounter

#Now automate the process of generating all the ACES for a given aptamer and
span of ACE lengths
def genACEs(aptName, aptSeq, minACElen = 8, maxACElen = 12):

    aptLen = len(aptSeq)
    aptRevComp = revComp(aptSeq)
    arraySize = numACEs(aptLen, minACElen, maxACElen)

    ACEarray = {'AptName': pd.Series('' for i in range(arraySize)),
                'AptSeq': pd.Series('' for i in range(arraySize)),
                'ACEnum': pd.Series(0 for i in range(arraySize)),
                'ACEname': pd.Series('' for i in range(arraySize)),
                'ACEseq': pd.Series('' for i in range(arraySize)),
                'ACElength': pd.Series('' for i in range(arraySize)),
                'ACEstartPos':  pd.Series(''  for  i  in  range(arraySize))
#Starting position along the 5'-3' aptamer sequence (5' apt = 0)
             }
    ACEframe = pd.DataFrame(data = ACEarray)

    counter = 0
    for n in range(minACElen, maxACElen+1):
        seqStart = aptLen-n
        for i in range(0, seqStart+1):
            start = seqStart-i

            ACEframe.at[counter, 'AptName'] = aptName
            ACEframe.at[counter, 'AptSeq'] = aptSeq
            ACEframe.at[counter, 'ACEnum'] = counter
```

```
            ACEframe.at[counter, 'ACEname'] = aptName + '_' + str(counter)

            ACEframe.at[counter, 'ACEseq'] = aptRevComp[start:start+n]

            ACEframe.at[counter, 'ACElength'] = len(aptRevComp[start:start+n])
            ACEframe.at[counter, 'ACEstartPos'] = int(i)

            counter = counter+1

    return ACEframe
```

```python
MN4seq = 'GGCGACAAGGAAAATCCTTCAACGAAGTGGGTCGCC'
MN6seq = 'GACAAGGAAAATCCTTCAACGAAGTGGGTC'
```

```python
#Make a data frame with all the aptamer names and their sequences
filepath = '/Users/isaac/Desktop/Lab_Projects/Mirkin_Lab_Projects/AFRL_C-ABN_Project/20190703_ArrayAnalysis/JorgeAptSequencesAndFlareList/20190708_FlaresAptamersSequences_Sorted.csv'
flaresAptamersSeqs = pd.read_csv(filepath, delimiter = ',')

#Now make an array containing all the aptamer names
aptSeqs = [flaresAptamersSeqs.AptamerName.unique(), flaresAptamersSeqs.AptamerSequence.unique()]
numApts = len(flaresAptamersSeqs.AptamerName.unique())

#Is there an easier way to do this?
flaresAptamersSeqs['uniqueAptNames'] = pd.Series('' for i in range(numApts))
flaresAptamersSeqs['uniqueAptSequences'] = pd.Series('' for i in range(numApts))

for i in range(numApts):
    flaresAptamersSeqs.at[i, 'uniqueAptNames'] = aptSeqs[0][i]
    flaresAptamersSeqs.at[i, 'uniqueAptSequences'] = aptSeqs[1][i]

uniqueApts = flaresAptamersSeqs[['uniqueAptNames', 'uniqueAptSequences']][0:61]
#uniqueApts.tail()
```

```python
uniqueApts.at[61, 'uniqueAptNames'] = 'MN4'
uniqueApts.at[61, 'uniqueAptSequences'] = MN4seq
uniqueApts.at[62, 'uniqueAptNames'] = 'MN6'
uniqueApts.at[62, 'uniqueAptSequences'] = MN6seq
```

```python
MN4_ACEs = genACEs(aptName = 'MN4',
                   aptSeq = returnAptSeq(aptName = 'MN4', aptArray =
uniqueApts))

MN4_ACEs = MN4_ACEs.drop(['ACElength', 'ACEstartPos'], axis=1)

MN4_ACEs.columns = ['Name', 'VariantNumber', 'ID', 'AptamerName',
'AptamerSequence']

MN6_ACEs = genACEs(aptName = 'MN6',
                   aptSeq = returnAptSeq(aptName = 'MN6', aptArray =
uniqueApts))

MN6_ACEs = MN6_ACEs.drop(['ACElength', 'ACEstartPos'], axis=1)

MN6_ACEs.columns = ['Name', 'VariantNumber', 'ID', 'AptamerName',
'AptamerSequence']

MN6_ACEs.tail()
```

```python
#Read in the list of flare names, features, etc
myACEarrayPath =
"/Users/isaac/Desktop/Lab_Projects/Mirkin_Lab_Projects/AFRL_C-
ABN_Project/20190703_ArrayAnalysis/JorgeAptSequencesAndFlareList/20190708_Fla
resAptamersSequences_Sorted.csv"
myACEarray = pd.read_csv(myACEarrayPath, delimiter = ',')

myACEarray.tail()
```

```python
#NOW, APPEND THE MN4_ACEs and MN6_ACEs data to myACEarray
myACEarray = myACEarray.append(MN4_ACEs)
myACEarray = myACEarray.append(MN6_ACEs)
myACEarray.tail()
```

```python
#Now also read in data from a single raw array
TestMicroarrayDataPath =
"/Users/isaac/Desktop/Lab_Projects/Mirkin_Lab_Projects/AFRL_C-
ABN_Project/20190703_ArrayAnalysis/20190520_1253_Hyb/SG17044572_258594510002_
S001_GE2 locbkrd 3x_1_1_PBS.txt"
TestMicroarrayData = pd.read_csv(TestMicroarrayDataPath, delimiter = '\t',
header = 9)

```

```python
```

```
#Now, need a way to filter the feature number, row, column, and probeName
arrayDataLayout = TestMicroarrayData.filter(['FeatureNum', 'Row', 'Col',
'ControlType', 'ProbeName'], axis = 1)
```

```python
#Want to bind the probeName in arrayDataLayout to the matching column values in
myACEarray
ACEarrayLength = myACEarray.shape[0]
myACEarray['ACEnumber'] = pd.Series(range(0,ACEarrayLength))
myACEarray = myACEarray.reset_index(drop = True)
myACEarray.tail()
```

```python
#myACEarray
```

```python
#Now, get ready to bind
arrayDataLayout['ID'] = pd.Series('')
arrayDataLayout['AptamerName'] = pd.Series('')
arrayDataLayout['VariantNumber'] = pd.Series(0)
arrayDataLayout['AptamerSequence'] = pd.Series('')
arrayDataLayout['ACEnumber'] = pd.Series(0)
```

```python
#Iterate through the whole layout array to bind the correct aptamer name and
sequence etc to the right features.
#Will take about 10 minutes for 62,000 features.
arrayDataLength = arrayDataLayout.shape[0]
startingIndex = arrayDataLayout.index.tolist()[0]
tic = time.clock()

for x in range(0, arrayDataLength):
    probeName = arrayDataLayout['ProbeName'][startingIndex + x]
    ACEarraySubset = myACEarray.loc[myACEarray['Name'] == probeName]
    nameIsPresent = ACEarraySubset.index.tolist() != []
    if nameIsPresent:
        ACEsubsetIndex = ACEarraySubset.index.tolist()[0]
        #print myACEarray.loc[ACEsubsetIndex, 'ID']
        arrayDataLayout.loc[startingIndex + x, 'ID'] =
myACEarray.loc[ACEsubsetIndex, 'ID']
        arrayDataLayout.loc[startingIndex + x, 'AptamerName'] =
myACEarray.loc[ACEsubsetIndex, 'AptamerName']
        arrayDataLayout.loc[startingIndex + x, 'VariantNumber'] =
myACEarray.loc[ACEsubsetIndex, 'VariantNumber']
```

```
        arrayDataLayout.loc[startingIndex + x, 'AptamerSequence'] =
myACEarray.loc[ACEsubsetIndex, 'AptamerSequence']
        arrayDataLayout.loc[startingIndex + x, 'ACEnumber'] =
myACEarray.loc[ACEsubsetIndex, 'ACEnumber']

toc = time.clock()
timespan = toc-tic
print timespan
```

```python
#Just export all the layout data for MN6 and MN4
MN6_Layout = arrayDataLayout[arrayDataLayout['AptamerName'] == 'MN6']
MN4_Layout = arrayDataLayout[arrayDataLayout['AptamerName'] == 'MN4']
MN6_Layout.to_csv("/Users/isaac/Desktop/Lab_Projects/Mirkin_Lab_Projects/AFRL
_C-
ABN_Project/20190703_ArrayAnalysis/JorgeAptSequencesAndFlareList/MN6_Layout.c
sv")
MN4_Layout.to_csv("/Users/isaac/Desktop/Lab_Projects/Mirkin_Lab_Projects/AFRL
_C-
ABN_Project/20190703_ArrayAnalysis/JorgeAptSequencesAndFlareList/MN4_Layout.c
sv")
```

```python
#I got the sequence hybridization deltaGs and self-complementary deltaGs for
all the flare/ACE sequences from Slava
#and Jorge. They used a commercial UNAfold software package (the melt.pl method)
to calculate them.
#Now, import that csv
deltaGsArrayPath =
"/Users/isaac/Desktop/Lab_Projects/Mirkin_Lab_Projects/AFRL_C-
ABN_Project/20190703_ArrayAnalysis/JorgeAptSequencesAndFlareList/20190710_Fla
res_DeltaGs.csv"
deltaGsArray = pd.read_csv(deltaGsArrayPath, delimiter = ',')
```

```python
#Now, the goal is to add a 'DeltaG' and 'Self dG' column to the larger array
arrayDataLayout['DeltaG'] = pd.Series()
arrayDataLayout['Self dG'] = pd.Series()
```

```python
#Now add DeltaG and Self dG values to the entire layout array
ArrayLayoutLength = arrayDataLayout.shape[0]
startingIndex = arrayDataLayout.index.tolist()[0]

tic = time.clock()
```

```
for x in range(0, ArrayLayoutLength):

    probeName = arrayDataLayout['ProbeName'][startingIndex + x]
    dGarraySubset = deltaGsArray.loc[deltaGsArray['Flares_ID'] == probeName]
#Returns an empty array if probeName isn't present in myACEarray
    nameIsPresent = dGarraySubset.index.tolist() != [] #Boolean that checks if
that probe name is present in myACEarray
    if nameIsPresent:
        ACEsubsetIndex = dGarraySubset.index.tolist()[0]
        arrayDataLayout.loc[startingIndex + x, 'DeltaG'] =
deltaGsArray.loc[ACEsubsetIndex, 'DeltaG']
        arrayDataLayout.loc[startingIndex + x, 'Self dG'] =
deltaGsArray.loc[ACEsubsetIndex, 'Self dG']

toc = time.clock()
timespan = toc-tic
print timespan
```

```python
#Good. Now, need to build a method to automate the process of extracting a
well's raw data,
#specifying the experiment conditions for that well's data,
#and appending all the relevant layout and experimental condition info.
def genLabeledMicroarrayData(dataFilePath,
                             layoutFilePath,
                             ExptStepIndex,
                             ExptStep,
                             Block,
                             BlockIndex,
                             BlockCondition,
                             BlockDHEASconc_uM,
                             DHEASaddedThisStep):

    #Generate data frames from the selected microarray data file and array
layout file
    arrayData = pd.read_csv(dataFilePath, delimiter = '\t', header = 9)
    print arrayData.iloc[0,13]
    arrayLayout = pd.read_csv(layoutFilePath, index_col=0)

    #Generate array to populate with experimental conditions
    ExptConditionArray = arrayLayout.filter('FeatureNum')

    #Get length of the layout array
    arrayLength = len(arrayLayout['FeatureNum'])

    ##Populate the Experimental Condition Array with the appropriate columns
```

```
    ExptConditionArray['ExptStepIndex'] = pd.Series(ExptStepIndex for i in
range(arrayLength))
    ExptConditionArray['ExptStep'] = pd.Series(ExptStep for i in
range(arrayLength))
    ExptConditionArray['Block'] = pd.Series(Block for i in range(arrayLength))
    ExptConditionArray['BlockIndex'] = pd.Series(BlockIndex for i in
range(arrayLength))
    ExptConditionArray['BlockCondition'] = pd.Series(BlockCondition for i in
range(arrayLength))
    ExptConditionArray['BlockDHEASconc_uM'] = pd.Series(BlockDHEASconc_uM for
i in range(arrayLength))
    ExptConditionArray['DHEASaddedThisStep'] = pd.Series(DHEASaddedThisStep
for i in range(arrayLength))

    #Join the ExptConditionArray to the layout array
    joinedLayoutArray = ExptConditionArray.join(arrayLayout, lsuffix = '',
rsuffix = '_redundant')

    #Now join the joinedLayoutArray to the data array, appending 'redundant' to
any columns with redundant names
    joinedDataArray =
joinedLayoutArray.set_index('FeatureNum').join(arrayData.set_index('FeatureNu
m'),
                                                   lsuffix =
'', rsuffix = '_redundant')
    #Now list all the redundant column names
    colsToDrop = [col for col in joinedDataArray.columns if 'redundant' in col]
    #Now use that list to remove all the redundant columns
    joinedDataArray = joinedDataArray.drop(colsToDrop, axis=1)

    print joinedDataArray.iloc[0, 4]

    #Finally, return the joinedDataArray
    return joinedDataArray
```

```python
#Now, the question is, it just wasting time to construct a method that automates
the extraction process?
#I don't think so.
#This function (extractLabeledSlideData) can take as inputs a list to be filled
with data frames,
#a folder path and the experimental conditions, and then run
genLabeledMicroarrayData

def extractLabeledSlideData(folder,
                            ExptStepIndex,
                            ExptStep,
```

```
                            TargetAddedThisStep,
                            pathToFolders                                   =
"/Users/isaac/Desktop/Lab_Projects/Mirkin_Lab_Projects/AFRL_C-
ABN_Project/20190703_ArrayAnalysis/",
                            constantPartOfFileName                          =
"SG17044572_258594510002_S001_GE2 locbkrd 3x_",
                            wellNameList = ['1_1_PBS', '1_2_300D', '1_3_300D',
'1_4_300D', '2_1_PBS', '2_2_30D', '2_3_30D', '2_4_30D'],
                            suffix = '.txt',
                            layoutPath                                      =
"/Users/isaac/Desktop/Lab_Projects/Mirkin_Lab_Projects/AFRL_C-
ABN_Project/20190703_ArrayAnalysis/20190718_arrayDataLayout.csv",
                            BlockList = ['1_1', '1_2', '1_3', '1_4', '2_1',
'2_2', '2_3', '2_4'],
                            BlockIndexList = [1, 2, 3, 4, 5, 6, 7, 8],
                            BlockConditionList = ['PBS Only', '300 uM DHEAS',
'300 uM DHEAS', '300 uM DHEAS',
                     'PBS Only', '30 uM DHEAS', '30 uM DHEAS', '30 uM DHEAS'],
                            BlockTargetConcList = [0, 300, 300, 300, 0, 30, 30,
30]):

    arrayList = [[],[],[],[],[],[],[],[]]

    for i in range(0, len(arrayList)):
        arrayPath = pathToFolders + folder + constantPartOfFileName +
wellNameList[i] + suffix
        arrayList[i] = genLabeledMicroarrayData(dataFilePath = arrayPath,
                                                layoutFilePath = layoutPath,
                                                ExptStepIndex = ExptStepIndex,
                                                ExptStep = ExptStep,
                                                DHEASaddedThisStep          =
TargetAddedThisStep,
                                                Block = BlockList[i],
                                                BlockIndex                  =
BlockIndexList[i],
                                                BlockCondition              =
BlockConditionList[i],
                                                BlockDHEASconc_uM           =
BlockTargetConcList[i])

    return arrayList

```

```python
#Now make it into a function that compiles/appends together array data from
different wells/experiments
def compileArraysData(arraysList, removeRedundantIndex = False):
    compiledArrays = arraysList[0]
    for i in range(1,len(arraysList)):
```

```
        compiledArrays = compiledArrays.append(arraysList[i])
    compiledArrays = compiledArrays.reset_index()
    if(removeRedundantIndex):
        compiledArrays = compiledArrays.drop(['index'], axis = 1)
    return compiledArrays
```

```python
#Now, can quickly extract and label all the data from all the wells and all the steps
step1_hybArrays = extractLabeledSlideData(folder = '20190520_1253_Hyb/',
                                          ExptStepIndex = 1,
                                          ExptStep = 'Hyb',
                                          TargetAddedThisStep = False)

step2_washArrays          =          extractLabeledSlideData(folder          = '20190520_1417_HybAndPBSwash/',
                                           ExptStepIndex = 2,
                                           ExptStep = 'PBS Wash',
                                           TargetAddedThisStep = False)

step3_10minDHEASarrays        =        extractLabeledSlideData(folder        = '20190520_1516_DHEAS10minInc/',
                                                   ExptStepIndex = 3,
                                                   ExptStep  =  '10  min  DHEAS Incubation',
                                                   TargetAddedThisStep = True)

step4_overnightDHEASarrays       =       extractLabeledSlideData(folder       = '20190521_0949_DHEAS_OvernightInc/',
                                                   ExptStepIndex = 4,
                                                   ExptStep  =  'Overnight  DHEAS Incubation',
                                                   TargetAddedThisStep = True)

```

```python
#Compile all the data from the different experiment steps into their own data frames
fullHybSlide = compileArraysData(step1_hybArrays)
fullWashSlide = compileArraysData(step2_washArrays)
full10minIncSlide = compileArraysData(step3_10minDHEASarrays)
fullOvernightIncSlide = compileArraysData(step4_overnightDHEASarrays)
```
```python
#Now make an array of those data frames...
```

```
fullExptArray = [fullHybSlide, fullWashSlide, full10minIncSlide, fullOvernightIncSlide]
```

```python
#...And compile them all together to generate the final, fully synthesized data set
tic = time.clock()
allExptData = compileArraysData(fullExptArray, removeRedundantIndex = True)
toc = time.clock()
print toc - tic
```

```python
#Now, let's write that compiled data set to a .csv file so it can be easily retrieved
tic = time.clock()

filePath = '/Users/isaac/Desktop/Lab_Projects/Mirkin_Lab_Projects/AFRL_C-ABN_Project/20190703_ArrayAnalysis/'
allExptData.to_csv(path_or_buf = filePath + '20190730_allExptData.csv')

toc = time.clock()
print toc - tic
```

```python
#Excellent, I've refactored the data processing and consolidation steps into something much easier to read
#and replicate. Now actually move on to analysis!
```

Code C4. Representative microarray data analysis markdown file.

```python
#Import relevant software packages
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.font_manager as font_manager
from matplotlib.lines import Line2D
from matplotlib.pyplot import figure
from matplotlib import colors
import time
```

```python
#Now, import the full 20191009 all expt data .csv file
exptArrayPath = '/Users/isaac/Desktop/Lab_Projects/Mirkin_Lab_Projects/AFRL_C-ABN_Project/20191008_Isaac_ArrayData/20191009_allExptData.csv'

allExptData = pd.read_csv(exptArrayPath, delimiter = ',')

```

```python
allExptData.head()
allExptData.tail()
allExptData.iloc[0:5, 3:25]
```

```python
#Pull out the relevant data for Dopa2 and for a negative control aptamer I didn't use--let's say Dopa130_169
Dopa2_all = allExptData[allExptData['ProbeName'].str.contains('Dopa2')]

```

```python
Dopa2_all.info()

#Note: not a lot of features with the Dopa130 name! Because a lot BE9th aptamers with similar sequences.
#Will need to find the alternatively named flare sequences and add them to the data frame
```

```python
Dopa2_all.AptamerSequence.values[0]
```

```python
#Make a data frame with all the aptamer names and their sequences

#NOTE: When Monica finishes what she's doing, can use her frame here
filepath = '/Users/isaac/Desktop/Lab_Projects/Mirkin_Lab_Projects/AFRL_C-ABN_Project/20191008_Isaac_ArrayData/JorgeAptSequencesAndFlareList/20191009_FlaresAptamersSequences_Updated.csv'
flaresAptamersSeqs = pd.read_csv(filepath, delimiter = ',')

#Now make an array containing all the aptamer names
aptSeqs = [flaresAptamersSeqs.AptamerName.unique(), flaresAptamersSeqs.AptamerSequence.unique()]
numApts = len(flaresAptamersSeqs.AptamerName.unique())

#Is there an easier way to do this?
flaresAptamersSeqs['uniqueAptNames'] = pd.Series('' for i in range(numApts))
flaresAptamersSeqs['uniqueAptSequences'] = pd.Series('' for i in range(numApts))

for i in range(numApts):
    flaresAptamersSeqs.at[i, 'uniqueAptNames'] = aptSeqs[0][i]
    flaresAptamersSeqs.at[i, 'uniqueAptSequences'] = aptSeqs[1][i]

uniqueApts = flaresAptamersSeqs[['uniqueAptNames', 'uniqueAptSequences']]
uniqueApts.iloc[50:80, 0:]

uniqueApts.at[69, 'uniqueAptNames'] = 'Dopa2'
uniqueApts.at[69, 'uniqueAptSequences'] = 'GGACGTGGATTTTCCGCATACGAAGTTGTCC'

```

```python
#Now I want a way to generate the reverse complement of all the aptamer sequences. Do that.
def revComp(seq):
    seqLength = len(seq)
    #print seqLength
    revSeq = seq[::-1]
    #print 'revSeq = ' + revSeq
    #print 'len(revSeq) = ' + str(len(revSeq))
    revCompSeq = ''
    for i in range(0, seqLength):
        currentBase = revSeq[i]
        if(currentBase == 'A' or currentBase == 'a'):
            revCompSeq = revCompSeq + 'T'
        if(currentBase == 'T' or currentBase == 't'):
            revCompSeq = revCompSeq + 'A'
```

```
        if(currentBase == 'G' or currentBase == 'g'):
            revCompSeq = revCompSeq + 'C'
        if(currentBase == 'C' or currentBase == 'c'):
            revCompSeq = revCompSeq + 'G'
    return revCompSeq

#Also make a way to generate the reverse complement of an RNA aptamer, in DNA
def revComp_RNAtoDNA(seq):
    seqLength = len(seq)
    #print seqLength
    revSeq = seq[::-1]
    #print 'revSeq = ' + revSeq
    #print 'len(revSeq) = ' + str(len(revSeq))
    revCompSeq = ''
    for i in range(0, seqLength):
        currentBase = revSeq[i]
        if(currentBase == 'A' or currentBase == 'a'):
            revCompSeq = revCompSeq + 'T'
        if(currentBase == 'U' or currentBase == 'u'):
            revCompSeq = revCompSeq + 'A'
        if(currentBase == 'G' or currentBase == 'g'):
            revCompSeq = revCompSeq + 'C'
        if(currentBase == 'C' or currentBase == 'c'):
            revCompSeq = revCompSeq + 'G'
    return revCompSeq

#Let's just make a quick function to make it quicker and easier to return the
sequence of an aptamer
#From the aptamer array
def returnAptSeq(aptName, aptArray = uniqueApts):
    aptSeq          =          aptArray[aptArray['uniqueAptNames']          ==
aptName].iloc[0]['uniqueAptSequences']
    return aptSeq

#Great, now make a quick method to automatically generate the number of ACEs,
given an aptamer length, a minimum
#ACE length, and a maximum ACE length
def numACEs(aptLen, minACElen, maxACElen):
    ACEcounter = 0
    for i in range(minACElen, maxACElen+1):
        ACEcounter = ACEcounter + aptLen - i + 1
    return ACEcounter

#Now automate the process of generating all the ACES for a given aptamer and
span of ACE lengths
def genACEs(aptName, aptSeq, minACElen = 8, maxACElen = 12):

    aptLen = len(aptSeq)
```

```
    aptRevComp = revComp(aptSeq)
    arraySize = numACEs(aptLen, minACElen, maxACElen)

    ACEarray = {'AptName': pd.Series('' for i in range(arraySize)),
                'AptSeq': pd.Series('' for i in range(arraySize)),
                'ACEnum': pd.Series(0 for i in range(arraySize)),
                'ACEname': pd.Series('' for i in range(arraySize)),
                'ACEseq': pd.Series('' for i in range(arraySize)),
                'ACElength': pd.Series('' for i in range(arraySize)),
                'ACEstartPos':  pd.Series(''  for  i  in  range(arraySize))
#Starting position along the 5'-3' aptamer sequence (5' apt = 0)
             }
    ACEframe = pd.DataFrame(data = ACEarray)

    counter = 0
    for n in range(minACElen, maxACElen+1):
        seqStart = aptLen-n
        for i in range(0, seqStart+1):
            start = seqStart-i

            ACEframe.at[counter, 'AptName'] = aptName
            ACEframe.at[counter, 'AptSeq'] = aptSeq
            ACEframe.at[counter, 'ACEnum'] = counter
            ACEframe.at[counter, 'ACEname'] = aptName + '_' + str(counter)

            ACEframe.at[counter, 'ACEseq'] = aptRevComp[start:start+n]

            ACEframe.at[counter, 'ACElength'] = len(aptRevComp[start:start+n])
            ACEframe.at[counter, 'ACEstartPos'] = int(i)

            counter = counter+1

    return ACEframe
```

```python
#See if the above methods work to generate ACE arrays for Dopa2 and Dopa130
Dopa2_ACEs = genACEs(aptName = 'Dopa2',
                         aptSeq  =  returnAptSeq(aptName  =  'Dopa2',  aptArray  =
uniqueApts))

```

```python
#Check that it worked:
Dopa2_ACEs['AptSeq'][0]
```

```
```

```python
#Here's a method for returning all the ACEs for an aptamer that are missing/named differently on the array
#Note: be careful not to run this method with ACE frames for different aptamers.
def genMissingACEframe(generatedACEframe, dataACEframe):
    #Want to verify that generatedACEframe and dataACEframe are for the same aptamer
    genACEaptName = generatedACEframe.AptName[0]
    dataACEaptName = dataACEframe.AptamerName.unique()[0] #Note that the dataACEframes have a different aptName vs aptamerName
    print 'Just to check, the generated ACE frame is for the aptamer ' + genACEaptName + '.'
    print 'Just to check, the data ACE frame is for the aptamer ' + dataACEaptName + '.'
    num_genACEs = len(generatedACEframe['ACEnum'])

    dataACEframe_varNums = dataACEframe.VariantNumber.unique()

    dataACEframe_varNumsInts = []
    #Now figure out how to fill in the gaps
    for i in range(0, len(dataACEframe_varNums)):
        dataACEframe_varNumsInts.append(int(dataACEframe_varNums[i]))

    missingVarList = []
    for i in range(0, num_genACEs):
        if(i not in dataACEframe_varNumsInts):
            missingVarList.append(i)

    numMissingACEs = len(missingVarList)

    print 'Generated ACE frame has ' + str(num_genACEs) + ' ACEs.'
    print 'Data ACE frame has ' + str(num_genACEs - numMissingACEs) + ' ACEs.'
    print 'Data ACE frame is missing ' + str(numMissingACEs) + ' ACEs compared to generated ACE frame;'
    print 'Return data frame containing all the ACEs in generated ACE frame missing from data ACE frame.'
    missingACEframe = generatedACEframe.loc[generatedACEframe['ACEnum'].isin(missingVarList)]

    return missingACEframe
```

```python

Dopa2_missingACEframe = genMissingACEframe(generatedACEframe = Dopa2_ACEs,
                                           dataACEframe = Dopa2_all)
```

```
Dopa2_missingACEframe.head()
```

```python
#Method to get a list of ACE sequences from a data frame containing an 'ACEseq'
column
def getACEseqList(ACEframe):
    num_ACEs = len(ACEframe.ACEseq)

    ACEseqList = []

    #Reindex in case the data frame has missing ACEs or is a subset of a
different data frame
    ACEframe_RI = ACEframe.reset_index()

    for i in range(0, num_ACEs):
        ACEseqList.append(ACEframe_RI.ACEseq[i])

    return ACEseqList
```

```python
#Use method to generate a list of the missing/alternatively named ACEs
Dopa2_missingACElist = Dopa2_missingACEframe.ACEseq.values
Dopa2_missingACElist
```

```python
#Method for returning all the data from a list of ACE names generated by
getACEseqList method
def getDataWithACElist(ACEseqList, dataframe):
    dataWithACEs = dataframe.loc[dataframe['ID'].isin(ACEseqList)]
    return dataWithACEs
```

```python
#Return all the data from the missing/alternatively named ACEs on the array
Dopa2_missingACEdata = getDataWithACElist(ACEseqList = Dopa2_missingACElist,
                                          dataframe = allExptData)
```

```python
#Method for re-labeling ACEs that bind to more than one aptamer, to make it
easier to analyze all ACEs that bind
#to a particular aptamer
def annotateMissingACEs(ACElistFrame, dataACEframe):
```

```
    arrayDataLength = dataACEframe.shape[0]

    dataACEframe_RI = dataACEframe.reset_index()

    tic = time.clock()

    for x in dataACEframe.index:
        seqACE = dataACEframe['ID'][x]

        relevantRow = ACElistFrame[ACElistFrame['ACEseq'] == seqACE]

        relevantAptName = relevantRow.iloc[0]['AptName']
        relevantAptSeq = relevantRow.iloc[0]['AptSeq']
        relevantACEnum = float(relevantRow.iloc[0]['ACEnum'])
        relevantACEname       =       relevantAptName       +       '_'       +
str(relevantRow.iloc[0]['ACEnum'])

        dataACEframe.at[x, 'ProbeName'] = relevantACEname
        dataACEframe.at[x, 'AptamerName'] = relevantAptName
        dataACEframe.at[x, 'AptamerSequence'] = relevantAptSeq
        dataACEframe.at[x, 'VariantNumber'] = relevantACEnum


    toc = time.clock()
    timespan = toc-tic

    print timespan
    return dataACEframe
```

```python
#Now in the data frame, annotate all the alternatively named ACEs.
Dopa2_missingACEdataLabeled = annotateMissingACEs(ACElistFrame = Dopa2_ACEs,
                                                  dataACEframe           =
Dopa2_missingACEdata)

```

```python
Dopa2_missingACEdataLabeled.head()
```

```python
#Now I want to append the data that was originally labeled with the aptamer
name,
#To the data that has just been reannotated with the chosen aptamer name
Dopa2_combinedData = Dopa2_all.append(Dopa2_missingACEdataLabeled)
```

```
```

```python
#Check dimensions of arrays to make sure they combined properly
Dopa2_all.info()
#Dopa2_combinedData.info()
#Dopa130_all.info()
#Dopa130_combinedData.info()
#Another check: see what the unique variant numbers are. Should be 0.0 - 154.0
#Dopa130_combinedData.sort_values(by=['VariantNumber']).VariantNumber.unique(
)
```

```python
#Convert this to a method:
#Method for adding ACE length and start position information to the data array
def addACElenAndStartPos(dataArray, ACEarray):
    counter = 0

    arrayLen = len(dataArray.ID)

    dataArray['ACElen'] = pd.Series(0 for i in range(arrayLen))
    dataArray['ACEstartPos'] = pd.Series(0 for i in range(arrayLen))


    for i in dataArray.index.values[:]:
        #print i
        thisACEseq = dataArray.at[i, 'ID']

        #print str(thisACEseq)

        thisACEinfo = ACEarray[ACEarray['ACEseq'] == thisACEseq]

        #print thisACEinfo

        thisACEindex = thisACEinfo.index.values[0]

        #print thisACEindex

        if(type(thisACEseq) == str):
            thisACElen = len(str(thisACEseq)) #Note that this might cooerce
some 'nan' values to str
            #print str(thisACElen)
            thisACEstartPos = thisACEinfo.ACEstartPos.values[0]
            #print str(thisACEstartPos)
            #print 'For index ' + str(i) + ', thisACEstartPos is a float.
```

```
            if(type(thisACEstartPos) == int):
                dataArray.at[i, 'ACElen'] = thisACElen
                dataArray.at[i, 'ACEstartPos'] = thisACEstartPos
            else:
                counter = counter+1
        else:
            'For index ' + str(i) + ', thisACEseq is not a str. It is a ' +
str(type(thisACEseq)) + ' with value ' + str(thisACEseq)

    print str(counter) + ' rows were not properly processed to add ACElen and
ACEstartPos values'

    return dataArray
```

```python
#Now, it might be good to sort by Expt condition, then by block, then by
variantNumber
Dopa2_sorted       =       Dopa2_combinedData.sort_values(by=['ExptStepIndex',
'BlockIndex', 'VariantNumber'])
Dopa2_sorted = Dopa2_sorted.reset_index(drop=True)
Dopa2_sorted.iloc[1000:1020, 5:25]

```

```python
#Now add ACE length and start position to the data frames
Dopa2_sorted = addACElenAndStartPos(Dopa2_sorted, Dopa2_ACEs)
```

```python
#Good, that worked. Now just need to export that data set to a csv file for
safe keeping and easy retrieval.
!mkdir '20191008_Isaac_ArrayData/20191014_Dopa2_dataAndHeatMaps/'
folderPath   =   '/Users/isaac/Desktop/Lab_Projects/Mirkin_Lab_Projects/AFRL_C-
ABN_Project/20191008_Isaac_ArrayData/20191014_Dopa2_dataAndHeatMaps/'
Dopa2_sorted.to_csv(path_or_buf = folderPath + '20191014_Dopa2_rawData.csv')
```

```python
#Method that cleans the data set, and prints out useful info as it does so.
#This method would do something like, for any given Agilent microarray dataset,
count the number of data points
#that have nonuniform features, nonuniform background, and have outlier
features/background values relative to
#replicates. Then the function should return an array with all those problematic
data points removed, and print
#how many data points were removed and the new dimensions of the array.
```

```
def cleanAgilentArray(arrayToClean):
    arrayShape = arrayToClean.shape
    numSpots = arrayShape[0]
    print str(numSpots) + ' feature spots in raw array.'

    #print 'Checking array for spots with nonuniform feature values:'
    nonUnifFeatArray = arrayToClean[arrayToClean['gIsFeatNonUnifOL'] == 1]
    nonUnifFeatShape = nonUnifFeatArray.shape
    numNonUnifFeat = nonUnifFeatShape[0]
    print str(numNonUnifFeat) + ' spots with nonuniform features.'

    #print 'Checking array for spots with nonuniform backgrounds:'
    nonUnifBGArray = arrayToClean[arrayToClean['gIsBGNonUnifOL'] == 1]
    nonUnifBGShape = nonUnifBGArray.shape
    numNonUnifBG = nonUnifBGShape[0]
    print str(numNonUnifBG) + ' spots with nonuniform backgrounds.'

    print 'Checking array for spots with outlier feature values:'
    OLfeatureArray = arrayToClean[arrayToClean['gIsFeatPopnOL'] == 1]
    OLfeatureArrayShape = OLfeatureArray.shape
    numOLfeatures = OLfeatureArrayShape[0]
    print str(numOLfeatures) + ' spots with values that are outliers relative
to other replicates in the same well.'

    print 'Checking array for spots with outlier background values:'
    OLBGArray = arrayToClean[arrayToClean['gIsBGPopnOL'] == 1]
    OLBGArrayShape = OLBGArray.shape
    numOLBGs = OLBGArrayShape[0]
    print str(numOLBGs) + ' spots with values that are outliers relative to
other replicates in the same well.'

    #print 'Now clean array by removing nonuniform and outlier spots:'
    cleanArray = arrayToClean[arrayToClean['gIsFeatNonUnifOL'] == 0]
    cleanArray = cleanArray[cleanArray['gIsBGNonUnifOL'] == 0]
    cleanArray = cleanArray[cleanArray['gIsFeatPopnOL'] == 0]
    cleanArray = cleanArray[cleanArray['gIsBGPopnOL'] == 0]

    cleanArrayShape = cleanArray.shape
    numCleanSpots = cleanArrayShape[0]
    numBadSpots = numSpots - numCleanSpots
    print ''
    print 'Cleaned array has ' + str(numCleanSpots) + ' spots, with ' +
str(numBadSpots) + ' bad spots removed.'
    print ''

    return cleanArray
```

```python
#Let's remove all the bad data points
cleanDopa2 = cleanAgilentArray(Dopa2_sorted)
```

```python
#Reset the indices for the cleaned arrays
cleanDopa2_RI = cleanDopa2.reset_index(drop = True)
```

```python
#Make an array of the names of the statistics columns to add
statColsToAdd = ['mean_gProcessedSignal',
                 'med_gProcessedSignal',
                 'std_gProcessedSignal',
                 'max_gProcessedSignal',
                 'min_gProcessedSignal',
                 'numReplicates',
                 'mean_rProcessedSignal',
                 'med_rProcessedSignal',
                 'std_rProcessedSignal',
                 'max_rProcessedSignal',
                 'min_rProcessedSignal']
```

```python
#Now add all the columns that will be required for calculating statistics
#Make this a method, actually
def addLabeledColumns(array, colsToAdd):
    arrayLen = len(array.iloc[:][array.columns[0]])

    numColumns = len(colsToAdd)

    for i in range(numColumns):
        array[colsToAdd[i]] = pd.Series(float('nan') for i in range(arrayLen))

    return array
```

```python
cleanDopa2 = addLabeledColumns(cleanDopa2_RI, statColsToAdd)

cleanDopa2.tail()
```

```python
```

```
#Make a function out of generating a single row of descriptive statistics
#Takes as input a frame to which the stat row will be appended, and the array with all the replicates
#to be statistically analyzed
def genStatRow(receivingFrame, arraySubSet):

    arraySubSetFirstIndex = arraySubSet.index[0]

    subSetFirstRow = arraySubSet[arraySubSet.index == arraySubSetFirstIndex]

    receivingFrame = receivingFrame.append(subSetFirstRow)

    lastRowIndex = receivingFrame.index[len(receivingFrame.index) - 1]

    mean_gProcessedSignal = arraySubSet['gProcessedSignal'].mean()
    receivingFrame.at[lastRowIndex,         'mean_gProcessedSignal']         = mean_gProcessedSignal

    median_gProcessedSignal = arraySubSet['gProcessedSignal'].median()
    receivingFrame.at[lastRowIndex,          'med_gProcessedSignal']         = median_gProcessedSignal

    std_gProcessedSignal = arraySubSet['gProcessedSignal'].std()
    receivingFrame.at[lastRowIndex,          'std_gProcessedSignal']         = std_gProcessedSignal

    max_gProcessedSignal = arraySubSet['gProcessedSignal'].max()
    receivingFrame.at[lastRowIndex,          'max_gProcessedSignal']         = max_gProcessedSignal

    min_gProcessedSignal = arraySubSet['gProcessedSignal'].min()
    receivingFrame.at[lastRowIndex,          'min_gProcessedSignal']         = min_gProcessedSignal

    numReps = len(arraySubSet['gProcessedSignal'])
    receivingFrame.at[lastRowIndex, 'numReplicates'] = numReps

    mean_rProcessedSignal = arraySubSet['rProcessedSignal'].mean()
    receivingFrame.at[lastRowIndex,         'mean_rProcessedSignal']         = mean_rProcessedSignal

    median_rProcessedSignal = arraySubSet['rProcessedSignal'].median()
    receivingFrame.at[lastRowIndex,          'med_rProcessedSignal']         = median_rProcessedSignal

    std_rProcessedSignal = arraySubSet['rProcessedSignal'].std()
    receivingFrame.at[lastRowIndex,          'std_rProcessedSignal']         = std_rProcessedSignal
```

```
    max_rProcessedSignal = arraySubSet['rProcessedSignal'].max()
    receivingFrame.at[lastRowIndex,         'max_rProcessedSignal']         =
max_rProcessedSignal

    min_rProcessedSignal = arraySubSet['rProcessedSignal'].min()
    receivingFrame.at[lastRowIndex,         'min_rProcessedSignal']         =
min_rProcessedSignal

    return receivingFrame
```

```python
#Now make a method for generating all the stats for all the ACE variants a
single block/subarray
def genStatBlock(receivingFrame, blockArray):

    numACEvariants = len(blockArray.VariantNumber.unique())

    for i in range(0, numACEvariants):
        arraySubSet = blockArray[blockArray['VariantNumber'] == float(i)]

        receivingFrame   =   genStatRow(receivingFrame   =   receivingFrame,
arraySubSet = arraySubSet)

    return receivingFrame
```

```python
#Now make a method for generating statistics for all the ACE variants in an
entire slide
def genSlideStats(receivingFrame, slideArray):
    numSubArrays = len(slideArray.BlockIndex.unique())

    for i in range(numSubArrays+1):
        subArray = slideArray[slideArray['BlockIndex'] == i]


        receivingFrame   =   genStatBlock(receivingFrame   =   receivingFrame,
blockArray = subArray)

    return receivingFrame
```

```python
#Now write a method to generate all the stats from a processed data array
def genExptStats(ExptDataArray):
```

```
    tic = time.clock()

    numExptSteps = len(ExptDataArray.ExptStepIndex.unique())

    recFrame = ExptDataArray[ExptDataArray.index == -1]

    for i in range(numExptSteps+1):
        slideArray = ExptDataArray[ExptDataArray['ExptStepIndex'] == i]

        recFrame = genSlideStats(receivingFrame = recFrame, slideArray =
slideArray)

    toc = time.clock()

    print toc-tic

    return recFrame
```

```python
#Now generate the stats
Dopa2_cleanedStats = genExptStats(cleanDopa2)
```

```python
#Great! Now save both as csv files.
folderPath = '/Users/isaac/Desktop/Lab_Projects/Mirkin_Lab_Projects/AFRL_C-
ABN_Project/20191008_Isaac_ArrayData/20191014_Dopa2_dataAndHeatMaps/'
Dopa2_cleanedStats.to_csv(path_or_buf = folderPath +
'20191014_Dopa2_cleanedStats.csv')


```

```python
Dopa2_cleanedStats = pd.read_csv(folderPath +
'20191014_Dopa2_cleanedStats.csv', delimiter = ',')
Dopa2_cleanedStats.tail()

```

```python
#divide the data frame by experiment step

Dopa2stats_1Hyb1 = Dopa2_cleanedStats[Dopa2_cleanedStats['ExptStepIndex'] ==
1]
```

```
Dopa2stats_2Hyb2 = Dopa2_cleanedStats[Dopa2_cleanedStats['ExptStepIndex'] == 2]
Dopa2stats_3DandC1hrInc = Dopa2_cleanedStats[Dopa2_cleanedStats['ExptStepIndex'] == 3]
Dopa2stats_4otherTargets1hrInc = Dopa2_cleanedStats[Dopa2_cleanedStats['ExptStepIndex'] == 4]



Dopa2stats_1Hyb1.tail()
#Dopa2stats_1Hyb.tail()
#Dopa130stats_4IncON.tail()
```


```python
#Now, before we move further, let's take a detour into quality control, with the bright spots!
#Can I retrieve the GE_brightSpotValues?
brightSpotFrame = allExptData[allExptData['ProbeName'] == 'GE_BrightCorner']
#Okay, now get the Bright Spot data fully processed
brightSpotFrame_clean = cleanAgilentArray(brightSpotFrame)
#Now sort the Bright Spot Data
brightSpotFrame_cleanSorted = brightSpotFrame_clean.sort_values(by=['ExptStepIndex', 'BlockIndex'])
#Now add stat columns
brightSpotsClean_preStats = addLabeledColumns(brightSpotFrame_cleanSorted, statColsToAdd)


#Now generate the stats. Need a custom For loop for this; can't iterate over the brightspots' VariantNumber
brightSpotStatsFrame = brightSpotsClean_preStats[brightSpotsClean_preStats['ExptStepIndex'] == -1]

ExptStepIndices = brightSpotsClean_preStats.ExptStepIndex.unique()

BlockIndices = brightSpotsClean_preStats.BlockIndex.unique()

for x in ExptStepIndices:
    for i in BlockIndices:
        condition1 = brightSpotsClean_preStats['ExptStepIndex'] == x
        condition2 = brightSpotsClean_preStats['BlockIndex'] == i

        slideArray = brightSpotsClean_preStats[condition1]

        blockArray = slideArray[condition2]
```

```
        brightSpotStatsFrame = genStatRow(brightSpotStatsFrame, blockArray)
        #print receivingFrame



#Now save the stats
folderPath  =  '/Users/isaac/Desktop/Lab_Projects/Mirkin_Lab_Projects/AFRL_C-
ABN_Project/20191008_Isaac_ArrayData/'
brightSpotStatsFrame.to_csv(path_or_buf         =         folderPath         +
'20191009_brightSpotsPosCtrls_cleanedStats.csv')
#Now retrieve the saved stats
brightSpotStatsFrame           =           pd.read_csv(folderPath           +
'20191009_brightSpotsPosCtrls_cleanedStats.csv')

brightSpotStatsFrame.tail()
```

```python
#Now split up by experimental step
brightSpotStats_1Hyb1                                                        =
brightSpotStatsFrame[brightSpotStatsFrame['ExptStepIndex'] == 1]
brightSpotStats_2Hyb2                                                        =
brightSpotStatsFrame[brightSpotStatsFrame['ExptStepIndex'] == 2]
brightSpotStats_3DandC1hrInc                                                 =
brightSpotStatsFrame[brightSpotStatsFrame['ExptStepIndex'] == 3]
brightSpotStats_4otherTargets1hrInc                                          =
brightSpotStatsFrame[brightSpotStatsFrame['ExptStepIndex'] == 4]

```

```python
#Now, get ready to plot some information about the bright spots--their average,
standard deviation, min and max
def plotStats(xAxis,
              yMeans,
              ySTDs,
              yMedians,
              yMaxes,
              yMins,
              title,
              xLab,
              yLab,
              date,
              figName,

              show = True,
              color = 'b',
              yMin = None,
```

```
            yMax = None,
            legendLoc = 'upper left'):
    # Set the font properties (for use in legend and the axes)
    font_path = '/library/fonts/Arial Bold.ttf'
    font_prop = font_manager.FontProperties(fname=font_path, size=14)
    font = {'family' : 'Arial',
            'weight' : 'bold',
            'size'   : 12}
    plt.rc('font', **font)

    #Define the figure
    fig, ax = plt.subplots(figsize=(7, 5), dpi=120)
    meansIm = ax.errorbar(xAxis,
                    yMeans,
                    ySTDs,
                    linestyle='None',
                    color = color,
                    marker='D')

    mediansIM = ax.scatter(xAxis,
                    yMedians,
                    color = 'c',
                    marker='D')

    maxesIm = ax.scatter(xAxis,
                    yMaxes,
                    color = 'r',
                    marker='D')

    minsIm = ax.scatter(xAxis,
                    yMins,
                    color = 'g',
                    marker='D')

    #Make a legend to describe the statistical elements
    ColorsStylesList = [color, 'c', 'r', 'g']
    legend_list = ['Mean+StdDev', 'Median', 'Maximum', 'Minimum']

    legend_elements = [{} for i in range(len(legend_list))]
    for i in range(len(legend_elements)):
        legend_elements[i] = Line2D([0], [0], color=ColorsStylesList[i], lw=0,
label=legend_list[i], marker = 'D')

    ax.legend(legend_elements,     legend_list,     loc     =     legendLoc,
prop={'family':'Arial', 'size':12, 'weight':'bold'})
```

```
    ax.set_title(title, fontname = 'Arial', fontsize = 16, fontweight = 'bold')

    ylab = yLab
    xlab = xLab

    ax.set_xlabel(xlab, fontproperties = font_prop)
    ax.set_ylabel(ylab, fontproperties = font_prop)

    if(yMin != None):
        ax.set_ylim(yMin, yMax)

    fig.tight_layout()

    #save figure
    if(show):
        plt.savefig(folderPath + date +'_' + figName)
        plt.show()
```

```python
#Try plotting bright spots stats for first experiment's first experiment step
fPath     =     '/Users/isaac/Desktop/Lab_Projects/Mirkin_Lab_Projects/AFRL_C-
ABN_Project/20191008_Isaac_ArrayData/20191009_brightSpotStatsFigures/'

plotStats(xAxis = brightSpotStats_1Hyb1['BlockIndex'],
                  yMeans = brightSpotStats_1Hyb1['mean_gProcessedSignal'],
                  ySTDs = brightSpotStats_1Hyb1['std_gProcessedSignal'],
                  yMedians = brightSpotStats_1Hyb1['med_gProcessedSignal'],
                  yMaxes = brightSpotStats_1Hyb1['max_gProcessedSignal'],
                  yMins = brightSpotStats_1Hyb1['min_gProcessedSignal'],
                  title = "Cy3 Bright Spot Fluorescence vs Subarray Index, 1Hyb1
Step",
                  xLab = 'Sub-Array Index',
                  yLab = 'Cy3 Fluorescence signal (AU)',
                  date = '20191009',
                  figName = 'brightSpotStats_1Hyb1_byBlockIndex',
                  color = 'k',
                  yMin = 0,
                  yMax = None,
                  folderPath = fPath)

plotStats(xAxis = brightSpotStats_2Hyb2['BlockIndex'],
                  yMeans = brightSpotStats_2Hyb2['mean_gProcessedSignal'],
                  ySTDs = brightSpotStats_2Hyb2['std_gProcessedSignal'],
                  yMedians = brightSpotStats_2Hyb2['med_gProcessedSignal'],
```

```
                  yMaxes = brightSpotStats_2Hyb2['max_gProcessedSignal'],
                  yMins = brightSpotStats_2Hyb2['min_gProcessedSignal'],
                  title = "Cy3 Bright Spot Fluorescence vs Subarray Index, 2Hyb2
Step",
                  xLab = 'Sub-Array Index',
                  yLab = 'Cy3 Fluorescence signal (AU)',
                  date = '20191009',
                  figName = 'brightSpotStats_2Hyb2_byBlockIndex',
                  color = 'k',
                  yMin = 0,
                  yMax = None,
                  folderPath = fPath)

plotStats(xAxis = brightSpotStats_3DandC1hrInc['BlockIndex'],
                  yMeans                                                    =
brightSpotStats_3DandC1hrInc['mean_gProcessedSignal'],
                  ySTDs                                                     =
brightSpotStats_3DandC1hrInc['std_gProcessedSignal'],
                  yMedians                                                  =
brightSpotStats_3DandC1hrInc['med_gProcessedSignal'],
                  yMaxes                                                    =
brightSpotStats_3DandC1hrInc['max_gProcessedSignal'],
                  yMins                                                     =
brightSpotStats_3DandC1hrInc['min_gProcessedSignal'],
                  title = "Cy3 Bright Spot Fluorescence vs Subarray Index,
3DandC1hrInc Step",
                  xLab = 'Sub-Array Index',
                  yLab = 'Cy3 Fluorescence signal (AU)',
                  date = '20191009',
                  figName = 'brightSpotStats_3DandC1hrInc_byBlockIndex',
                  color = 'k',
                  yMin = 0,
                  yMax = None,
                  folderPath = fPath)

plotStats(xAxis = brightSpotStats_4otherTargets1hrInc['BlockIndex'],
                  yMeans                                                    =
brightSpotStats_4otherTargets1hrInc['mean_gProcessedSignal'],
                  ySTDs                                                     =
brightSpotStats_4otherTargets1hrInc['std_gProcessedSignal'],
                  yMedians                                                  =
brightSpotStats_4otherTargets1hrInc['med_gProcessedSignal'],
                  yMaxes                                                    =
brightSpotStats_4otherTargets1hrInc['max_gProcessedSignal'],
                  yMins                                                     =
brightSpotStats_4otherTargets1hrInc['min_gProcessedSignal'],
                  title = "Cy3 Bright Spot Fluorescence vs Subarray Index,
4otherTargets1hrInc Step",
                  xLab = 'Sub-Array Index',
```

```
                  yLab = 'Cy3 Fluorescence signal (AU)',
                  date = '20191009',
                  figName                                                 =
'brightSpotStats_4otherTargets1hrInc_byBlockIndex',
                  color = 'k',
                  yMin = 0,
                  yMax = None,
                  folderPath = fPath,
                  legendLoc = 'upper center')
```

```python
#Note: the bright spots don't fluoresce in the Cy5 channel
```

```python
#Cool. So tomorrow, plot by experiment step
brightSpotStats_Block_1_1                                                 =
brightSpotStatsFrame[brightSpotStatsFrame['BlockIndex'] == 1]
brightSpotStats_Block_1_2                                                 =
brightSpotStatsFrame[brightSpotStatsFrame['BlockIndex'] == 2]
brightSpotStats_Block_1_3                                                 =
brightSpotStatsFrame[brightSpotStatsFrame['BlockIndex'] == 3]
brightSpotStats_Block_1_4                                                 =
brightSpotStatsFrame[brightSpotStatsFrame['BlockIndex'] == 4]
brightSpotStats_Block_2_1                                                 =
brightSpotStatsFrame[brightSpotStatsFrame['BlockIndex'] == 5]
brightSpotStats_Block_2_2                                                 =
brightSpotStatsFrame[brightSpotStatsFrame['BlockIndex'] == 6]
brightSpotStats_Block_2_3                                                 =
brightSpotStatsFrame[brightSpotStatsFrame['BlockIndex'] == 7]
brightSpotStats_Block_2_4                                                 =
brightSpotStatsFrame[brightSpotStatsFrame['BlockIndex'] == 8]
```

```python
#Try plotting bright spots stats for first experiment's first experiment step
plotStats(xAxis = brightSpotStats_Block_1_1['ExptStepIndex'],
                  yMeans = brightSpotStats_Block_1_1['mean_gProcessedSignal'],
                  ySTDs = brightSpotStats_Block_1_1['std_gProcessedSignal'],
                  yMedians                                                =
brightSpotStats_Block_1_1['med_gProcessedSignal'],
                  yMaxes = brightSpotStats_Block_1_1['max_gProcessedSignal'],
                  yMins = brightSpotStats_Block_1_1['min_gProcessedSignal'],
                  title = "Cy3 Bright Spot Fluorescence vs Expt Step, Subarray
1_1",
                  xLab = 'Sub-Array Index',
                  yLab = 'Cy3 Fluorescence signal (AU)',
                  date = '20191009',
```

```
                figName = 'brightSpotStats_Subarray_1_1_byExptStep',
                color = 'k',
                yMin = 0,
                yMax = None,
                folderPath = fPath,
                legendLoc = 'lower left')

plotStats(xAxis = brightSpotStats_Block_1_2['ExptStepIndex'],
                yMeans = brightSpotStats_Block_1_2['mean_gProcessedSignal'],
                ySTDs = brightSpotStats_Block_1_2['std_gProcessedSignal'],
                yMedians                                                    =
brightSpotStats_Block_1_2['med_gProcessedSignal'],
                yMaxes = brightSpotStats_Block_1_2['max_gProcessedSignal'],
                yMins = brightSpotStats_Block_1_2['min_gProcessedSignal'],
                title = "Cy3 Bright Spot Fluorescence vs Expt Step, Subarray
1_2",
                xLab = 'Sub-Array Index',
                yLab = 'Cy3 Fluorescence signal (AU)',
                date = '20191009',
                figName = 'brightSpotStats_Subarray_1_2_byExptStep',
                color = 'k',
                yMin = 0,
                yMax = None,
                folderPath = fPath)

plotStats(xAxis = brightSpotStats_Block_1_3['ExptStepIndex'],
                yMeans = brightSpotStats_Block_1_3['mean_gProcessedSignal'],
                ySTDs = brightSpotStats_Block_1_3['std_gProcessedSignal'],
                yMedians                                                    =
brightSpotStats_Block_1_3['med_gProcessedSignal'],
                yMaxes = brightSpotStats_Block_1_3['max_gProcessedSignal'],
                yMins = brightSpotStats_Block_1_3['min_gProcessedSignal'],
                title = "Cy3 Bright Spot Fluorescence vs Expt Step, Subarray
1_3",
                xLab = 'Sub-Array Index',
                yLab = 'Cy3 Fluorescence signal (AU)',
                date = '20191009',
                figName = 'brightSpotStats_Subarray_1_3_byExptStep',
                color = 'k',
                yMin = 0,
                yMax = None,
                folderPath = fPath,
                legendLoc = 'lower left')

plotStats(xAxis = brightSpotStats_Block_1_4['ExptStepIndex'],
                yMeans = brightSpotStats_Block_1_4['mean_gProcessedSignal'],
                ySTDs = brightSpotStats_Block_1_4['std_gProcessedSignal'],
```

```
                yMedians                                                =
brightSpotStats_Block_1_4['med_gProcessedSignal'],
                yMaxes = brightSpotStats_Block_1_4['max_gProcessedSignal'],
                yMins = brightSpotStats_Block_1_4['min_gProcessedSignal'],
                title = "Cy3 Bright Spot Fluorescence vs Expt Step, Subarray
1_4",
                xLab = 'Sub-Array Index',
                yLab = 'Cy3 Fluorescence signal (AU)',
                date = '20191009',
                figName = 'brightSpotStats_Subarray_1_4_byExptStep',
                color = 'k',
                yMin = 0,
                yMax = None,
                folderPath = fPath)

plotStats(xAxis = brightSpotStats_Block_2_1['ExptStepIndex'],
                yMeans = brightSpotStats_Block_2_1['mean_gProcessedSignal'],
                ySTDs = brightSpotStats_Block_2_1['std_gProcessedSignal'],
                yMedians                                                =
brightSpotStats_Block_2_1['med_gProcessedSignal'],
                yMaxes = brightSpotStats_Block_2_1['max_gProcessedSignal'],
                yMins = brightSpotStats_Block_2_1['min_gProcessedSignal'],
                title = "Cy3 Bright Spot Fluorescence vs Expt Step, Subarray
2_1",
                xLab = 'Sub-Array Index',
                yLab = 'Cy3 Fluorescence signal (AU)',
                date = '20191009',
                figName = 'brightSpotStats_Subarray_2_1_byExptStep',
                color = 'k',
                yMin = 0,
                yMax = None,
                folderPath = fPath,
                legendLoc = 'lower left')

plotStats(xAxis = brightSpotStats_Block_2_2['ExptStepIndex'],
                yMeans = brightSpotStats_Block_2_2['mean_gProcessedSignal'],
                ySTDs = brightSpotStats_Block_2_2['std_gProcessedSignal'],
                yMedians                                                =
brightSpotStats_Block_2_2['med_gProcessedSignal'],
                yMaxes = brightSpotStats_Block_2_2['max_gProcessedSignal'],
                yMins = brightSpotStats_Block_2_2['min_gProcessedSignal'],
                title = "Cy3 Bright Spot Fluorescence vs Expt Step, Subarray
2_2",
                xLab = 'Sub-Array Index',
                yLab = 'Cy3 Fluorescence signal (AU)',
                date = '20191009',
                figName = 'brightSpotStats_Subarray_2_2_byExptStep',
                color = 'k',
```

```
                  yMin = 0,
                  yMax = None,
                  folderPath = fPath)

plotStats(xAxis = brightSpotStats_Block_2_3['ExptStepIndex'],
                  yMeans = brightSpotStats_Block_2_3['mean_gProcessedSignal'],
                  ySTDs = brightSpotStats_Block_2_3['std_gProcessedSignal'],
                  yMedians                                                    =
brightSpotStats_Block_2_3['med_gProcessedSignal'],
                  yMaxes = brightSpotStats_Block_2_3['max_gProcessedSignal'],
                  yMins = brightSpotStats_Block_2_3['min_gProcessedSignal'],
                  title = "Cy3 Bright Spot Fluorescence vs Expt Step, Subarray
2_3",
                  xLab = 'Sub-Array Index',
                  yLab = 'Cy3 Fluorescence signal (AU)',
                  date = '20191009',
                  figName = 'brightSpotStats_Subarray_2_3_byExptStep',
                  color = 'k',
                  yMin = 0,
                  yMax = None,
                  folderPath = fPath)

plotStats(xAxis = brightSpotStats_Block_2_4['ExptStepIndex'],
                  yMeans = brightSpotStats_Block_2_4['mean_gProcessedSignal'],
                  ySTDs = brightSpotStats_Block_2_4['std_gProcessedSignal'],
                  yMedians                                                    =
brightSpotStats_Block_2_4['med_gProcessedSignal'],
                  yMaxes = brightSpotStats_Block_2_4['max_gProcessedSignal'],
                  yMins = brightSpotStats_Block_2_4['min_gProcessedSignal'],
                  title = "Cy3 Bright Spot Fluorescence vs Expt Step, Subarray
2_4",
                  xLab = 'Sub-Array Index',
                  yLab = 'Cy3 Fluorescence signal (AU)',
                  date = '20191009',
                  figName = 'brightSpotStats_Subarray_2_4_byExptStep',
                  color = 'k',
                  yMin = 0,
                  yMax = None,
                  folderPath = fPath)
```

```python
#This is a function that returns a data frame with the average and standard
deviation within an array,
#grouped by a selected category, and returning a single, particular column
def returnAvgAndStd(array, groupByThisCat, returnThisColumn):
    catUniques = array[groupByThisCat].unique()
    #print catUniques
```

```
    #print groupByThisCat
    catLen = len(catUniques)
    #print catLen

    avgDataName = 'Avg' + returnThisColumn
    stdDataName = 'Std' + returnThisColumn

    catAvg = [float(i) for i in np.zeros(catLen)]
    catStd = [float(i) for i in np.zeros(catLen)]

    d = {groupByThisCat: catUniques, avgDataName: catAvg, stdDataName: catStd}
    processedDF = pd.DataFrame(data=d).reset_index()
    processedDF    =    processedDF[['index',    groupByThisCat,    avgDataName,
stdDataName]]

    #processedData = pd.DataFrame(data = [catUniques, catAvg, catStd])


    for i in range(catLen):
        processedDF.at[i,   avgDataName]   =   array[array[groupByThisCat]   ==
catUniques[i]][returnThisColumn].mean()

        #If for any reason there's only one number here, don't get std this way
        #if(len(array[array[groupByThisCat]                                    ==
catUniques[i]][returnThisColumn]) > 1):
        processedDF.at[i,   stdDataName]   =   array[array[groupByThisCat]   ==
catUniques[i]][returnThisColumn].std()
        #else:

    #processedData = [catUniques, catAvg, catStd]
    return processedDF
```


```python
#All right, see if I can generate heatmaps for Dopa2 and for the Dopa130 negative
control
#Function for making a data frame that can/will be converted into a heat map
def genFrameForHeatmap(array, returnThisCol):
    ACElens = array['ACElen'].unique()
    shortestACElen = ACElens[0]

    outputFrame = returnAvgAndStd(array[array['ACElen'] == shortestACElen],
                                                        'ACEstartPos',
                                                        returnThisCol)
    outputFrame['ACElen']    =    pd.Series(8    for    i    in    range(0,
len(outputFrame['ACEstartPos'])))
```

```
    aptSeq = array['AptamerSequence'].values[0]
    outputFrame['AptamerSequence'] = pd.Series(aptSeq for i in range(0,
len(outputFrame['ACEstartPos'])))

    aptName = array['AptamerName'].values[0]
    outputFrame['AptamerName'] = pd.Series(aptName for i in range(0,
len(outputFrame['ACEstartPos'])))

    for i in ACElens[1:]:
        dummyFrame = returnAvgAndStd(array[array['ACElen'] == i],
                                             'ACEstartPos',
                                             returnThisCol)
        dummyFrame['ACElen'] = pd.Series(i for x in range(0,
len(dummyFrame['ACEstartPos'])))
        dummyFrame['AptamerSequence'] = pd.Series(aptSeq for i in range(0,
len(dummyFrame['ACEstartPos'])))
        dummyFrame['AptamerName'] = pd.Series(aptName for i in range(0,
len(outputFrame['ACEstartPos'])))

        outputFrame = outputFrame.append(dummyFrame)
    return outputFrame
```

```python
#Let's try generating a heatmap for each individual sub
Dopa2_AvgHybHeatMap_1Hyb1_1_1 = genFrameForHeatmap(
    array = Dopa2stats_1Hyb1[Dopa2stats_1Hyb1['BlockIndex'] == 1],
    returnThisCol = 'gProcessedSignal')

```

```python
Dopa2_AvgHybHeatMap_1Hyb1_1_1.tail()
```

```python
#Function for generating/extracting the heatmap data frame to an array before
plotting it
def genHeatMapArray(heatMapFrame, returnThisColumn = 'rProcessedSignal'):
    ACElens = heatMapFrame.ACElen.unique()

    heatMapArray = []

    for x in ACElens:
        heatMapFrame_xbp = heatMapFrame[
        heatMapFrame['ACElen'] == x][returnThisColumn].values
```

```
        heatMapFrame_xbp = np.append(
            heatMapFrame_xbp, np.repeat(np.nan, x-1)).tolist()

        heatMapArray.append(heatMapFrame_xbp)

    return heatMapArray
```

```python
#Function for actually plotting the heatmap
def heatMap(FrameForHeatMap, columnToPlot, title, colorBarLabel, fontSize, colorMax, colorMin, figName, date, cbar_kw={},
              folderPath = '/Users/isaac/Desktop/Lab_Projects/Mirkin_Lab_Projects/AFRL_C-ABN_Project/20191008_Isaac_ArrayData/'):

    #Array data
    heatMapArray = genHeatMapArray(FrameForHeatMap, returnThisColumn = 'Avg' + columnToPlot)
    #print heatMapArray
    #Two axes for the heatmap: ACE/flare length and ACE/flare position on aptamer
    ACElensFloats = set(FrameForHeatMap['ACElen'].values) #Make this the Y axis
    ACElens = [int(i) for i in ACElensFloats]

    AptSeq = FrameForHeatMap['AptamerSequence'].values[0]
    AptSeq_allCaps = AptSeq.upper()
    AptSeqList = list(AptSeq_allCaps[i] for i in range(len(AptSeq))) #Make this the X axis

    # Set the font dictionaries (for plot title and axis titles)
    title_font = {'fontname':'Arial', 'size':str(fontSize), 'color':'black', 'weight':'bold',
                  'verticalalignment':'bottom'} # Bottom vertical alignment for more space
    axis_font = {'fontname':'Arial', 'size':str(fontSize), 'color':'black', 'weight':'bold'}


    # Set the font properties (for use in legend and the axes)
    font_path = '/library/fonts/Arial Bold.ttf'
    font_prop = font_manager.FontProperties(fname=font_path, size=str(fontSize))

    font = {'family' : 'Arial',
            'weight' : 'bold',
            'size'   : fontSize}

    plt.rc('font', **font)
```

```
    #Define the figure
    fig, ax = plt.subplots(figsize=(12, 3), dpi=120)
    im = ax.imshow(heatMapArray)


    # We want to show all ticks...
    ax.set_xticks(np.arange(len(AptSeqList)))
    ax.set_yticks(np.arange(len(ACElens)))

    # ... and label them with the respective list entries
    ax.set_xticklabels(AptSeqList)
    ax.set_yticklabels(ACElens)

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=0, ha="right",
         rotation_mode="anchor")

    ax.set_title(title, fontname = 'Arial', fontsize = fontSize+2, fontweight
= 'bold')

    aptName = FrameForHeatMap['AptamerName'].values[0]

    ylab = 'Flare Length, bp'
    xlab = 'Start of Flare Binding Position on ' + aptName + ' Aptamer'

    ax.set_xlabel(xlab, fontproperties = font_prop)
    ax.set_ylabel(ylab, fontproperties = font_prop)

    #Add color bar
    #SET THE RANGE OF THE COLORBAR
    norm = colors.Normalize(vmin=colorMin, vmax=colorMax)
    im.set_norm(norm)
    cbar = ax.figure.colorbar(im, ax=ax, **cbar_kw)
    cbar.ax.set_ylabel(colorBarLabel,        rotation=-90,        va="bottom",
fontproperties = font_prop)

    fig.tight_layout()
    #save figure
    plt.savefig(folderPath + date +'_' + figName)

    plt.show()
```


```python
#Plot an average of all the heatmaps from 1Hyb1
```

```
Dopa2_AvgHybHeatMap_1Hyb1allBlocksCy3 = genFrameForHeatmap(
    array = Dopa2stats_1Hyb1,
    returnThisCol = 'gProcessedSignal')

Dopa2_AvgHybHeatMap_1Hyb1allBlocksCy3_max                                    =
Dopa2_AvgHybHeatMap_1Hyb1allBlocksCy3.AvggProcessedSignal.max()


heatMap(FrameForHeatMap = Dopa2_AvgHybHeatMap_1Hyb1allBlocksCy3,
            columnToPlot = 'gProcessedSignal',
            title = 'DHEA-S Aptamer Flare Binding Profile',
            colorBarLabel = 'Cy5 Hyb Signal, AU',
            fontSize = 12,
            colorMin = 0,
            colorMax = Dopa2_AvgHybHeatMap_1Hyb1allBlocksCy3_max,
            figName = 'DIS11th_3_1Hyb1_heatMap',
            date = '20191014',
            cbar_kw = {},
            folderPath = folderPath)
```

```python
#Now make a method for generating all those individual heatmaps
def genHeatMapsBySubArray(slideStatsFrame,
                          titlePrefix,
                          colorBarLabel,
                          fontSize,
                          colorMin,
                          colorMax,
                          figNamePrefix,
                          date,
                          cbar_kw = {},
                          columnToPlot = 'gProcessedSignal',
                          folderPath                                         =
'/Users/isaac/Desktop/Lab_Projects/Mirkin_Lab_Projects/AFRL_C-
ABN_Project/20191008_Isaac_ArrayData/'):
    subArrayIndexes = slideStatsFrame.BlockIndex.unique()
    subArrayLabels = slideStatsFrame.Block.unique()

    for i in subArrayIndexes:
        slideStatsFrame_i = genFrameForHeatmap(
            array = slideStatsFrame[slideStatsFrame['BlockIndex'] == i],
            returnThisCol = columnToPlot)

        heatMap(FrameForHeatMap = slideStatsFrame_i,
            columnToPlot = columnToPlot,
            title = titlePrefix + ', Block ' + subArrayLabels[i-1],
```

```
            colorBarLabel = colorBarLabel,
            fontSize = fontSize,
            colorMin = colorMin,
            colorMax = colorMax,
            figName = figNamePrefix + '_Block_' + subArrayLabels[i-1],
            date = date,
            cbar_kw = {},
            folderPath = folderPath)
```

```python
#Okay, interesting. Get the exact same hybridization pattern for all the subarrays, but the absolute
#fluorescence values vary pretty significantly between the arrays. If we normalize the fluorescence values
#to the bright spot values, does that change/improve consistency of the values between arrays?
#Data frames to work from on normalization:
#Dopa2_cleanedStats
#Dopa130_cleanedStats
#brightSpotStatsFrame
```

```python
normColsToAdd = ['mean_gProcessedSignal_normToBSmean',
                 'med_gProcessedSignal_normToBSmean',
                 'std_gProcessedSignal_normToBSmean',
                 'max_gProcessedSignal_normToBSmean',
                 'min_gProcessedSignal_normToBSmean',
                 'mean_rProcessedSignal_normToBSmean',
                 'med_rProcessedSignal_normToBSmean',
                 'std_rProcessedSignal_normToBSmean',
                 'max_rProcessedSignal_normToBSmean',
                 'min_rProcessedSignal_normToBSmean',
                 'mean_gProcessedSignal_normToBSmedian',
                 'med_gProcessedSignal_normToBSmedian',
                 'std_gProcessedSignal_normToBSmedian',
                 'max_gProcessedSignal_normToBSmedian',
                 'min_gProcessedSignal_normToBSmedian',
                 'mean_rProcessedSignal_normToBSmedian',
                 'med_rProcessedSignal_normToBSmedian',
                 'std_rProcessedSignal_normToBSmedian',
                 'max_rProcessedSignal_normToBSmedian',
                 'min_rProcessedSignal_normToBSmedian']
```

```python
#Add the columns to the stat frames
Dopa2_cleanedStats_withNormToBS    =    addLabeledColumns(Dopa2_cleanedStats,
normColsToAdd)
Dopa2_cleanedStats_withNormToBS.tail()
```

```python
exptStepIndex = Dopa2_cleanedStats_withNormToBS.ExptStepIndex[0]
blockIndex = Dopa2_cleanedStats_withNormToBS.BlockIndex[0]
brightSpotFilter = (brightSpotStatsFrame['ExptStepIndex'] == exptStepIndex) &
(brightSpotStatsFrame['BlockIndex'] == exptStepIndex)
relevantBSdata = brightSpotStatsFrame[brightSpotFilter]
relevantBSdata
relevantBSmean = relevantBSdata.mean_gProcessedSignal[0]
relevantBSmedian = relevantBSdata.med_gProcessedSignal[0]
print relevantBSmean
print relevantBSmedian

#normalizedStatsFrame = Dopa2_cleanedStats_withNormToBS.copy()
#normalizedStatsFrame.head()
```

```
34250.970000000016
36013.10000000001
```

```python
#Now, iterate through and normalize to either the BS mean or the BS median
def addNormtoBS_inEachSubarray_stats(statsFrameToNormalize, BSstatsFrame):
    tic = time.clock()

    normalizedStatsFrame = statsFrameToNormalize.copy()

    for i in statsFrameToNormalize.index:
        #What I want to do here is to first, get the experiment step index and
the subarray index
        exptStepIndex = normalizedStatsFrame.ExptStepIndex[i]
        blockIndex = normalizedStatsFrame.BlockIndex[i]

        #Next, I want to retrieve the mean and median gProcessedSignal from the
brightSpotsStatsFrame
        brightSpotFilter = (BSstatsFrame['ExptStepIndex'] == exptStepIndex) &
(BSstatsFrame['BlockIndex'] == exptStepIndex)
        relevantBSdata = BSstatsFrame[brightSpotFilter]
```

```
        relevantBSmean = relevantBSdata.mean_gProcessedSignal.values[0]
        relevantBSmedian = relevantBSdata.med_gProcessedSignal.values[0]

        #Add the stats normalized to the BSmean
        #Green Channel
        normalizedStatsFrame.at[i,   'mean_gProcessedSignal_normToBSmean']   =
normalizedStatsFrame.mean_gProcessedSignal[i] / relevantBSmean
        normalizedStatsFrame.at[i,   'med_gProcessedSignal_normToBSmean']   =
normalizedStatsFrame.med_gProcessedSignal[i] / relevantBSmean
        normalizedStatsFrame.at[i,   'std_gProcessedSignal_normToBSmean']   =
normalizedStatsFrame.std_gProcessedSignal[i] / relevantBSmean
        normalizedStatsFrame.at[i,   'max_gProcessedSignal_normToBSmean']   =
normalizedStatsFrame.max_gProcessedSignal[i] / relevantBSmean
        normalizedStatsFrame.at[i,   'min_gProcessedSignal_normToBSmean']   =
normalizedStatsFrame.min_gProcessedSignal[i] / relevantBSmean
        #Red Channel (note that I'm normalizing the red channel by green channel
positive controls; unclear if this will work yet)
        normalizedStatsFrame.at[i,   'mean_rProcessedSignal_normToBSmean']   =
normalizedStatsFrame.mean_rProcessedSignal[i] / relevantBSmean
        normalizedStatsFrame.at[i,   'med_rProcessedSignal_normToBSmean']   =
normalizedStatsFrame.med_rProcessedSignal[i] / relevantBSmean
        normalizedStatsFrame.at[i,   'std_rProcessedSignal_normToBSmean']   =
normalizedStatsFrame.std_rProcessedSignal[i] / relevantBSmean
        normalizedStatsFrame.at[i,   'max_rProcessedSignal_normToBSmean']   =
normalizedStatsFrame.max_rProcessedSignal[i] / relevantBSmean
        normalizedStatsFrame.at[i,   'min_rProcessedSignal_normToBSmean']   =
normalizedStatsFrame.min_rProcessedSignal[i] / relevantBSmean

        #Add the stats normalized to the BSmedian
        #Green Channel
        normalizedStatsFrame.at[i,  'mean_gProcessedSignal_normToBSmedian']  =
normalizedStatsFrame.mean_gProcessedSignal[i] / relevantBSmedian
        normalizedStatsFrame.at[i,  'med_gProcessedSignal_normToBSmedian']  =
normalizedStatsFrame.med_gProcessedSignal[i] / relevantBSmedian
        normalizedStatsFrame.at[i,  'std_gProcessedSignal_normToBSmedian']  =
normalizedStatsFrame.std_gProcessedSignal[i] / relevantBSmedian
        normalizedStatsFrame.at[i,  'max_gProcessedSignal_normToBSmedian']  =
normalizedStatsFrame.max_gProcessedSignal[i] / relevantBSmedian
        normalizedStatsFrame.at[i,  'min_gProcessedSignal_normToBSmedian']  =
normalizedStatsFrame.min_gProcessedSignal[i] / relevantBSmedian
        #Red Channel (note that I'm normalizing the red channel by green channel
positive controls; unclear if this will work yet)
        normalizedStatsFrame.at[i,  'mean_rProcessedSignal_normToBSmedian']  =
normalizedStatsFrame.mean_rProcessedSignal[i] / relevantBSmedian
        normalizedStatsFrame.at[i,  'med_rProcessedSignal_normToBSmedian']  =
normalizedStatsFrame.med_rProcessedSignal[i] / relevantBSmedian
        normalizedStatsFrame.at[i,  'std_rProcessedSignal_normToBSmedian']  =
normalizedStatsFrame.std_rProcessedSignal[i] / relevantBSmedian
        normalizedStatsFrame.at[i,  'max_rProcessedSignal_normToBSmedian']  =
normalizedStatsFrame.max_rProcessedSignal[i] / relevantBSmedian
```

```
        normalizedStatsFrame.at[i,   'min_rProcessedSignal_normToBSmedian']   =
normalizedStatsFrame.min_rProcessedSignal[i] / relevantBSmedian

    toc = time.clock()
    print toc-tic
    return normalizedStatsFrame

```

```python
Dopa2_cleanedStats_withNormToBS                                              =
addNormtoBS_inEachSubarray_stats(statsFrameToNormalize                       =
Dopa2_cleanedStats_withNormToBS,
                                                          BSstatsFrame       =
brightSpotStatsFrame)

```

```
    28.09941

```

```python
Dopa2_cleanedStats_withNormToBS.head()

```

```python
#Cool, so the negative control aptamer mainly doesn't show fluorescence, except
for at a couple of locations
#Where it looks like, by chance, the flare sequences were able to bind to one
of the aptamers added on
```

```python
#allExptData[allExptData['ControlType'] == 1].SystematicName.unique()
#IMPORTANT QUESTION: WHAT DO ALL THE DIFFERENT CONTROL SPOTS DO?
#I've used GE_BrightCorner spots (brightSpots),
#But there's also:
controlFeatures  =  ['GE_BrightCorner',  'DarkCorner',  'E1A_r60_a22',  'ERCC-
00053_71',
       'ERCC-00062_278', 'ETG10_13482', 'ERCC-00012_90', 'ERCC-00077_121',
       'ERCC-00075_180', 'E1A_r60_a107', 'ETG08_142674', 'E1A_r60_a135',
       'ERCC-00104_60', 'ERCC-00171_229', 'ETG05_66023', 'ERCC-00097_63',
       'ERCC-00028_121', 'E1A_r60_3', 'ETG09_35454', 'E1A_r60_1',
       'ETG04_27747', 'ERCC-00043_129', 'E1A_r60_a97', 'ETG09_205211',
       'ERCC-00160_243', 'ETG10_195139', 'ETG05_36762', 'ETG09_48764',
       'E1A_r60_n9', 'E1A_r60_a104', 'E1A_r60_n11', 'ETG07_105829',
```

```
        'ERCC-00144_60', 'ETG02_36680', 'E1A_r60_a20', 'ETG10_236652',
        'ETG10_234183', 'DCP_20_9', 'DCP_22_0', 'DCP_22_9', 'DCP_22_6',
        'DCP_1_0', 'DCP_20_1', 'DCP_22_4', 'DCP_20_7', 'DCP_20_0',
        'DCP_20_3', 'DCP_1_4', 'DCP_1_11', 'DCP_22_7', 'DCP_1_2',
        'DCP_22_2', 'DCP_20_5', 'DCP_1_1', 'DCP_1_7']
    #What do they all do?
```

```python
flaresAptamersSeqs[['AptamerName', 'AptamerSequence']]
aptNamesAndSeqs = pd.DataFrame()
aptNamesAndSeqs['AptamerSequence'] = flaresAptamersSeqs.AptamerSequence.unique()
aptNamesAndSeqs['AptamerName'] = flaresAptamersSeqs.AptamerName.unique()

#aptNamesAndSeqs

#Maybe the extra Dopa aptamers are better negative controls, since they seem to have more divergent sequences
#than the other aptamers
#Try out running Dopa130 as the negative control
```

```python
#Now, next steps: generate raw fluorescence heatmaps for all subarrays for all experimental steps, in the green
#And the red channel, normalized to the bright spot fluorescence and not.

#Then, zoom in on one particular flare strand--the 12bp Dopa2 flare strand.
#Examine how its fluorescence changes as a function of subarray and experimental step.
#Also examine if/how these trends change when the fluorescence values are normalized to the bright spot.
#Then, calculate fold change in fluorescence relative to the previous experimental step for all sequences.
    #Do this with and without normalizing to bright spot fluorescence.
#Then, look at how fluorescence changes in the calibration subarray, with and without BrightSpot normalization
#Then, normalize all other changes in fluorescence by the calibration subarray, either with or without brightSpot
    #normalization
#Then, calculate the k_off rate for flares free in buffer, based on the normalized change in fluorescence divided
    #by 1 hour
#And calculate the k_off_IF rate for flares in the presence of target molecule
#And then plot heatmaps that show where k_off_IF is larger than k_off
```

```

```python
#Generate raw fluorescence heatmaps for all subarrays for all experimental steps, in the green
#And the red channel, normalized to the bright spot fluorescence and not.

```

```python
#I should abstract all this plotting to a function
#Take as input a list of frames, a list of title prefixes, a color bar label, a list of figure name prefixes,
#a date, colorbar keywords, columnToPlot, and folderPath to save files
def genHeatMapsBySubArrayForMultipleExptSteps(frameList, titlePrefixList, figNamePrefixList,
                                              colorBarLabel, fontSize, colorMin, colorMax,
                                              date, columnToPlot, cbar_kw = {},
                                             folderPath = '/Users/isaac/Desktop/Lab_Projects/Mirkin_Lab_Projects/AFRL_C-ABN_Project/20191008_Isaac_ArrayData/'):
    for i in range(len(frameList)):
        genHeatMapsBySubArray(slideStatsFrame = frameList[i],
                              titlePrefix = titlePrefixList[i],
                              colorBarLabel = colorBarLabel,
                              fontSize = fontSize,
                              colorMin = colorMin,
                              colorMax = colorMax,
                              figNamePrefix = figNamePrefixList[i],
                              date = date,
                              cbar_kw = cbar_kw,
                              columnToPlot = columnToPlot)

```

```python
#Dopa2 not normed to BS, green channel
Dopa2stats_frameList = [Dopa2stats_1Hyb1,
         Dopa2stats_2Hyb2,
         Dopa2stats_3DandC1hrInc,
         Dopa2stats_4otherTargets1hrInc]

Dopa2stats_titlePrefixList = ['Dopa2 Hyb, Cy3, Step 1Hyb1',
```
```

```
                                    'Dopa2 Hyb, Cy3, Step 2Hyb2',
                                    'Dopa2 Hyb, Cy3, Step 3DandC1hrInc',
                                    'Dopa2 Hyb, Cy3, Step 4otherTargets1hrInc']

Dopa2stats_figNamePrefixList = ['Dopa2_Cy3Hyb_Heatmap_Step_1Hyb1_Block_',
                                      'Dopa2_Cy3Hyb_Heatmap_Step_2Hyb2_Block_',

'Dopa2_Cy3Hyb_Heatmap_Step_3DandC1hrInc_Block_',

'Dopa2_Cy3Hyb_Heatmap_Step_4otherTargets1hrInc_Block_']

#Set colorMin and colorMax relative to the highest fluorescence on the first
scan
Dopa2stats_1Hyb1_greenMax = Dopa2stats_1Hyb1.gProcessedSignal.max()


colorBarLabel = 'Cy3 Hyb Signal, AU'
fontSize = 12
colorMin = 0
colorMax = Dopa2stats_1Hyb1_greenMax
date = '20191014'
columnToPlot = 'gProcessedSignal'
cbar_kw = {}
folderPath  =  '/Users/isaac/Desktop/Lab_Projects/Mirkin_Lab_Projects/AFRL_C-
ABN_Project/20191008_Isaac_ArrayData/'

genHeatMapsBySubArrayForMultipleExptSteps(frameList = Dopa2stats_frameList,
                                          titlePrefixList                    =
Dopa2stats_titlePrefixList,
                                          figNamePrefixList                  =
Dopa2stats_figNamePrefixList,
                                          colorBarLabel = colorBarLabel,
                                          fontSize = fontSize,
                                          colorMin = colorMin,
                                          colorMax = colorMax,
                                          date = date,
                                          columnToPlot = columnToPlot,
                                          cbar_kw = cbar_kw,
                                          folderPath = folderPath)

```

```python
#Dopa2 not normed to BS, red channel
Dopa2stats_frameList = [Dopa2stats_1Hyb1,
         Dopa2stats_2Hyb2,
         Dopa2stats_3DandC1hrInc,
         Dopa2stats_4otherTargets1hrInc]
```

```
Dopa2stats_titlePrefixList = ['Dopa2 Hyb, Cy5, Step 1Hyb1',
                              'Dopa2 Hyb, Cy5, Step 2Hyb2',
                              'Dopa2 Hyb, Cy5, Step 3DandC1hrInc',
                              'Dopa2 Hyb, Cy5, Step 4otherTargets1hrInc']

Dopa2stats_figNamePrefixList = ['Dopa2_Cy5Hyb_Heatmap_Step_1Hyb1_Block_',
                                'Dopa2_Cy5Hyb_Heatmap_Step_2Hyb2_Block_',

'Dopa2_Cy5Hyb_Heatmap_Step_3DandC1hrInc_Block_',

'Dopa2_Cy5Hyb_Heatmap_Step_4otherTargets1hrInc_Block_']

#Set colorMin and colorMax relative to the highest fluorescence on the first
scan
Dopa2stats_1Hyb1_redMax = Dopa2stats_1Hyb1.rProcessedSignal.max()

colorBarLabel = 'Cy5 Hyb Signal, AU'
fontSize = 12
colorMin = 0
colorMax = Dopa2stats_1Hyb1_redMax
date = '20191014'
columnToPlot = 'rProcessedSignal'
cbar_kw = {}
folderPath  =  '/Users/isaac/Desktop/Lab_Projects/Mirkin_Lab_Projects/AFRL_C-
ABN_Project/20191008_Isaac_ArrayData/'

genHeatMapsBySubArrayForMultipleExptSteps(frameList = Dopa2stats_frameList,
                                          titlePrefixList                   =
Dopa2stats_titlePrefixList,
                                          figNamePrefixList                 =
Dopa2stats_figNamePrefixList,
                                          colorBarLabel = colorBarLabel,
                                          fontSize = fontSize,
                                          colorMin = colorMin,
                                          colorMax = colorMax,
                                          date = date,
                                          columnToPlot = columnToPlot,
                                          cbar_kw = cbar_kw,
                                          folderPath = folderPath)
```

```python
#Dopa2 normed to BS, green channel
Dopa2_cleanedStats_withNormToBS_1Hyb1 = Dopa2_cleanedStats_withNormToBS[
    Dopa2_cleanedStats_withNormToBS['ExptStepIndex'] == 1]
Dopa2_cleanedStats_withNormToBS_2Hyb2 = Dopa2_cleanedStats_withNormToBS[
```

```
    Dopa2_cleanedStats_withNormToBS['ExptStepIndex'] == 2]
Dopa2_cleanedStats_withNormToBS_3DandC1hrInc                                    =
Dopa2_cleanedStats_withNormToBS[
    Dopa2_cleanedStats_withNormToBS['ExptStepIndex'] == 3]
Dopa2_cleanedStats_withNormToBS_4otherTargets1hrInc                             =
Dopa2_cleanedStats_withNormToBS[
    Dopa2_cleanedStats_withNormToBS['ExptStepIndex'] == 4]

Dopa2stats_normToBS_frameList = [Dopa2_cleanedStats_withNormToBS_1Hyb1,
         Dopa2_cleanedStats_withNormToBS_2Hyb2,
         Dopa2_cleanedStats_withNormToBS_3DandC1hrInc,
         Dopa2_cleanedStats_withNormToBS_4otherTargets1hrInc]

Dopa2stats_normToBS_titlePrefixList = ['Dopa2 Hyb Norm to BS, Cy3, Step 1Hyb1',
                                'Dopa2 Hyb Norm to BS, Cy3, Step 2Hyb2',
                                'Dopa2   Hyb   Norm   to   BS,   Cy3,   Step
3DandC1hrInc',
                                'Dopa2   Hyb   Norm   to   BS,   Cy3,   Step
4otherTargets1hrInc']

Dopa2stats_normToBS_figNamePrefixList                                           =
['Dopa2_Cy3Hyb_NormToBS_Heatmap_Step_1Hyb1_Block_',

'Dopa2_Cy3Hyb_NormToBS_Heatmap_Step_2Hyb2_Block_',

'Dopa2_Cy3Hyb_NormToBS_Heatmap_Step_3DandC1hrInc_Block_',

'Dopa2_Cy3Hyb_NormToBS_Heatmap_Step_4otherTargets1hrInc_Block_']

#Set colorMin and colorMax relative to the highest fluorescence on the first
scan
Dopa2_cleanedStats_withNormToBS_1Hyb1_greenMax                                  =
Dopa2_cleanedStats_withNormToBS_1Hyb1.mean_gProcessedSignal_normToBSmean.max(
)


colorBarLabel = 'Cy3 Hyb Signal, AU'
fontSize = 12
colorMin = 0
colorMax = Dopa2_cleanedStats_withNormToBS_1Hyb1_greenMax
date = '20191014'
columnToPlot = 'mean_gProcessedSignal_normToBSmean'
cbar_kw = {}
folderPath   =   '/Users/isaac/Desktop/Lab_Projects/Mirkin_Lab_Projects/AFRL_C-
ABN_Project/20191008_Isaac_ArrayData/'

genHeatMapsBySubArrayForMultipleExptSteps(frameList                             =
Dopa2stats_normToBS_frameList,
```

```
                                        titlePrefixList                 =
Dopa2stats_normToBS_titlePrefixList,
                                        figNamePrefixList               =
Dopa2stats_normToBS_figNamePrefixList,
                                        colorBarLabel = colorBarLabel,
                                        fontSize = 16,
                                        colorMin = colorMin,
                                        colorMax = colorMax,
                                        date = date,
                                        columnToPlot = columnToPlot,
                                        cbar_kw = cbar_kw,
                                        folderPath = folderPath)
```

```python
Dopa2_cleanedStats_withNormToBS_1Hyb1.head()
```

```python
#Now do the normalized Cy5 values
Dopa2stats_normToBS_titlePrefixList = ['Dopa2 Hyb Norm to BS, Cy5, Step 1Hyb1',
                                       'Dopa2 Hyb Norm to BS, Cy5, Step 2Hyb2',
                                       'Dopa2   Hyb   Norm   to   BS,   Cy5,   Step
3DandC1hrInc',
                                       'Dopa2   Hyb   Norm   to   BS,   Cy5,   Step
4otherTargets1hrInc']

Dopa2stats_normToBS_figNamePrefixList                                          =
['Dopa2_Cy5Hyb_NormToBS_Heatmap_Step_1Hyb1_Block_',

'Dopa2_Cy5Hyb_NormToBS_Heatmap_Step_2Hyb2_Block_',

'Dopa2_Cy5Hyb_NormToBS_Heatmap_Step_3DandC1hrInc_Block_',

'Dopa2_Cy5Hyb_NormToBS_Heatmap_Step_4otherTargets1hrInc_Block_']

#Set colorMin and colorMax relative to the highest fluorescence on the first
scan
Dopa2_cleanedStats_withNormToBS_1Hyb1_redMax                                   =
Dopa2_cleanedStats_withNormToBS_1Hyb1.mean_rProcessedSignal_normToBSmean.max(
)



colorBarLabel = 'Cy5 Hyb Signal, AU'
fontSize = 12
colorMin = 0
colorMax = Dopa2_cleanedStats_withNormToBS_1Hyb1_redMax
date = '20191014'
```

```
columnToPlot = 'mean_rProcessedSignal_normToBSmean'
cbar_kw = {}
folderPath   =   '/Users/isaac/Desktop/Lab_Projects/Mirkin_Lab_Projects/AFRL_C-ABN_Project/20191008_Isaac_ArrayData/'

genHeatMapsBySubArrayForMultipleExptSteps(frameList                          = Dopa2stats_normToBS_frameList,
                                          titlePrefixList                    = Dopa2stats_normToBS_titlePrefixList,
                                          figNamePrefixList                  = Dopa2stats_normToBS_figNamePrefixList,
                                          colorBarLabel = colorBarLabel,
                                          fontSize = fontSize,
                                          colorMin = colorMin,
                                          colorMax = colorMax,
                                          date = date,
                                          columnToPlot = columnToPlot,
                                          cbar_kw = cbar_kw,
                                          folderPath = folderPath)
```

```python
#Hmm, so what I seem to be seeing from this is that the average of bright corner values in a subarray
#Doesn't necessarily normalize fluorescence values consistently
#In particular, setting the color bar relative to the maximum hybridized 1Hyb1 value doesn't guarantee
#that heatmaps from later conditions won't have large numbers of sequences with higher normalized fluorescence
#values relative to their bright spots
#One question worth pursuing is whether these subarray-specific trends are consistent across multiple aptamers
#And this is a question I can pursue and answer fairly quickly, by copying this notebook and running the analysis
#for many other aptamers
```

```python
#Plot an average of all the heatmaps from 1Hyb1 in Cy3 and Cy5 channels
Dopa2_AvgHybHeatMap_1Hyb1_red = genFrameForHeatmap(
    array = Dopa2stats_1Hyb1,
    returnThisCol = 'rProcessedSignal')

colorMax_red = Dopa2_AvgHybHeatMap_1Hyb1_red.AvgrProcessedSignal.max()

Dopa2_AvgHybHeatMap_1Hyb1_green = genFrameForHeatmap(
```

```
    array = Dopa2stats_1Hyb1,
    returnThisCol = 'gProcessedSignal')

colorMax_green = Dopa2_AvgHybHeatMap_1Hyb1_green.AvggProcessedSignal.max()

heatMap(FrameForHeatMap = Dopa2_AvgHybHeatMap_1Hyb1_red,
            columnToPlot = 'rProcessedSignal',
            title = 'Dopa2 Aptamer Flare Binding Profile',
            colorBarLabel = 'Cy5 Hyb Signal, AU',
            fontSize = fontSize,
            colorMin = colorMin,
            colorMax = colorMax_red,
            figName = 'Dopa2_1Hyb1_AllCy5_heatMap',
            date = '20191014',
            cbar_kw = {},
            folderPath = folderPath)

heatMap(FrameForHeatMap = Dopa2_AvgHybHeatMap_1Hyb1_green,
            columnToPlot = 'gProcessedSignal',
            title = 'Dopa2 Aptamer Flare Binding Profile',
            colorBarLabel = 'Cy3 Hyb Signal, AU',
            fontSize = fontSize,
            colorMin = colorMin,
            colorMax = colorMax_green,
            figName = 'Dopa2_1Hyb1_AllCy3_heatMap',
            date = '20191014',
            cbar_kw = {},
            folderPath = folderPath)
```

APPENDIX D: Supplementary Code for Chapter 4

Code D1. Matlab script for quantifying CRISPR-mediated insertion/deletion mutations.

```
WTnuc='target genomic site sequence';
%WTnuc='complimentary sequence to the above target site sequence';
%cycle through fastq files for different samples
files=dir('*.fastq');
indelstart=69;
width=40;
flank=10;
SUMMARY={};
SUMMARY{1,1}='Filename';
SUMMARY{1,2}='Skipped reads';
SUMMARY{1,3}='not INDEL';
SUMMARY{1,4}='Insertions';
SUMMARY{1,5}='Deletions';
SUMMARY{1,6}='INDEL rate';
foldername=strcat(num2str(width),'_',num2str(indelstart),'summary.csv');

for d=1:total sample number
    filename=files(d).name;
    %read fastq file
    [header,seqs,qscore] = fastqread(filename);
    seqsLength = length(seqs);          % number of sequences
    seqsFile = strcat(strrep(filename,'.fastq',''),'_INDELS');      % trims
off .fastq
    %create a directory with the same name as fastq file+_INDELS
    if exist(seqsFile,'dir');
        error('Directory already exists. Please rename or move it before
moving on.');
    end
    mkdir(seqsFile);                    % make directory
    wtLength = length(WTnuc);           % length of wildtype sequence
    sBLength = length(seqs);            % number of sequences

    % initialize counters and cell arrays
    nSkips=0;
    notINDEL=0;
    ins={};
    dels={};
    NumIns=0;
    NumDels=0;
    % iterate through each sequencing read
    for i = 1:sBLength
    %search for 10BP sequences that should flank both sides of the "INDEL
WINDOW"
        windowstart=strfind(seqs{i},WTnuc(indelstart-flank:indelstart));
```

```
        windowend=strfind(seqs{i},WTnuc(indelstart+width:indelstart+width+fla
nk));
        %if these flanks are found proceed\
        if length(windowstart)==1 && length(windowend)==1
            %if the sequence length matches the INDEL window length save as
            %not INDEL
            if windowend-windowstart==width+flank
                notINDEL=notINDEL+1;
            %if the sequence is two or more baseslonger than the INDEL
            %window length save as an Insertion
            elseif windowend-windowstart>=width+flank+1
                NumIns=NumIns+1;
                ins{NumIns,2}=seqs{i};
                ins{NumIns,1}=filename(1:2);
            %if the sequence is two or more bases shorter than the INDEL
            %window length save as a Deletion and name the second column
            %with the fileame
            elseif windowend-windowstart<=width+flank-1
                NumDels=NumDels+1;
                dels{NumDels,2}=seqs{i};
                dels{NumDels,1}=filename(1:2);
            %keep track of skipped sequences that are either one base
            %shorter or longer than the INDEL window width
            else
                nSkips=nSkips+1;
            end
        %keep track of skipped sequences that do not posess matching flank
        %sequences
        else
            nSkips=nSkips+1;
        end
    end
    SUMMARY{d+1,1}=seqsFile;
    SUMMARY{d+1,2}=nSkips;
    SUMMARY{d+1,3}=notINDEL;
    SUMMARY{d+1,4}=NumIns;
    SUMMARY{d+1,5}=NumDels;
    SUMMARY{d+1,6}=(NumIns+NumDels)/(NumIns+NumDels+notINDEL);
    fid=fopen(strcat(seqsFile, '/summary.txt'), 'wt');
    fprintf(fid, 'Skipped reads %i\n not INDEL %i\n Insertions %i\n Deletions
%i\n', [nSkips, notINDEL, NumIns, NumDels]);
    fclose(fid);
    save(strcat(seqsFile, '/nSkips'), 'nSkips');
    save(strcat(seqsFile, '/notINDEL'), 'notINDEL');
    save(strcat(seqsFile, '/NumIns'), 'NumIns');
    save(strcat(seqsFile, '/NumDels'), 'NumDels');
    save(strcat(seqsFile, '/dels'), 'dels');
```

```
    dlmcell(strcat(seqsFile, strcat('/dels_', filename(1:2), '.csv')), dels,
',');
    save(strcat(seqsFile, '/ins'), 'ins');
    dlmcell(strcat(seqsFile, strcat('/ins_', filename(1:2), '.csv')), ins,
',');
end
dlmcell(strcat(foldername), SUMMARY, ',')
```