Analysis of the Security Design, Engineering, and Implementation of the SecureDNA System

Alan T. Sherman, Jeremy J. Romanik Romano,
Edward Zieglar, Enis Golaszewski, Jonathan D. Fuchs
Cyber Defense Lab
University of Maryland, Baltimore County (UMBC)
Baltimore, Maryland 21250
Email: {sherman,jeremyr2,eziegl1,golaszewski,jfuchs2}@umbc.edu

William E. Byrd
Hugh Kaul Precision Medicine Institute
Heersink School of Medicine
University of Alabama at Birmingham
Birmingham, Alabama 35294
Email:webyrd@uab.edu

Abstract—We analyze security aspects of the SecureDNA system regarding its system design, engineering, and implementation. This system enables DNA synthesizers to screen order requests against a database of hazards. By applying novel cryptography involving distributed oblivious pseudorandom functions, the system aims to keep order requests and the database of hazards secret. Discerning the detailed operation of the system in part from source code (Version 1.0.8), our analysis examines key management, certificate infrastructure, authentication, and rate-limiting mechanisms. We also perform the first formal-methods analysis of the mutual authentication, basic request, and exemption-handling protocols.

Without breaking the cryptography, our main finding is that SecureDNA's custom mutual authentication protocol SCEP achieves only one-way authentication: the hazards database and keyservers never learn with whom they communicate. This structural weakness violates the principle of defense in depth and enables an adversary to circumvent rate limits that protect the secrecy of the hazards database, if the synthesizer connects with a malicious or corrupted keyserver or hashed database. We point out an additional structural weakness that also violates the principle of defense in depth: inadequate cryptographic bindings prevent the system from detecting if responses, within a TLS channel, from the hazards database were modified. Consequently, if a synthesizer were to reconnect with the database over the same TLS session, an adversary could replay and swap responses from the database without breaking TLS. Although the SecureDNA implementation does not allow such reconnections, it would be stronger security engineering to avoid the underlying structural weakness. We identify these vulnerabilities and suggest and verify mitigations, including adding strong bindings. Software Version 1.1.0 fixes SCEP with our proposed SCEP+ protocol.

Our work illustrates that a secure system needs more than sound mathematical cryptography; it also requires formal specifications, sound key management, proper binding of protocol message components, and careful attention to engineering and implementation details.

I. INTRODUCTION

The combination of gene editing technology (e.g., CRISPR [1]), DNA synthesis, and AI [2] poses an existential threat to humanity. Advances now enable a single malicious actor, with modest resources, to synthesize pathogens capable of starting pandemics. As an initial response to this threat, an international group of eminent biologists and cryptographers designed and implemented the SecureDNA system [3], which provides a method for honest DNA synthesis labs to screen synthesis-order requests for known biohazards. In late 2022, the SecureDNA Foundation [4] launched the system.

SecureDNA is significant for its advanced technology and impactful policy. It is the first fully privacy-preserving, cryptographically verifiable screening system. It operates in near real time through a distributed network of independent authorities and supports standardized verifiable compliance. SecureDNA's long-term strategy is to push for regulation mandating universal screening of DNA synthesis orders. Unfortunately, currently a malicious entity could avoid SecureDNA controls by purchasing their own synthesis machine for less than $50,000, or by sending their request to a lab that does not screen.

Using a *distributed oblivious pseudorandom function (DOPRF)*, the system enables synthesizers to screen order requests against a curated database of hazards while keeping the database and requests secret [5], [6]. To eliminate a possible single critical point of failure, the system distributes the key used by the DOPRF among n keyservers using Shamir secret sharing [7], [8].

We analyze security aspects of the SecureDNA system, focusing on its system design, engineering, and implementation. As part of this study, we perform the first formal-methods symbolic analysis of the two main protocols in SecureDNA for *structural* (fundamental logical) weaknesses. These protocols are the *basic order-request protocol* and the *exemption-handling protocol*, which permits a qualifying customer to order certain dangerous sequences. Our work studies the security of the SecureDNA system design and implementation. We do not review the bio-design, high-level crypto-design, or low-level software security. We do not examine the non-public monitoring and auditing that play an important role in operational system security.

Because the available written descriptions do not adequately describe the protocols, we first discern these protocols in part by examining the source code (Version 1.0.8) [9]. We

also examine PCAP data from our working local copy of the system. We then model these protocols and precisely state security goals. Using the *Cryptographic Protocol Shapes Analyzer (CPSA)* [10], we analyze whether the models achieve the goals. Our analysis includes a careful examination of SecureDNA's key and certificate management systems and their use in authentication and rate-limiting of queries to the hazards database.

Despite rigorous *universal composability (UC) proofs* [11] of the abstract cryptography [5], the concrete protocols in use in the SecureDNA system lack detailed descriptions and formal models, and the written descriptions lack verifiable, precisely-stated security goals of the system. Thus, our starting point is largely the SecureDNA system's whitepaper [3] and publicly-released source code [9].

Experience shows that there is great value in formal-methods analysis of cryptographic protocols, including in the design phases, because humans are poorly suited to analyze their complex nuanced security properties and their many possible execution sequences. For example, in 1995, using a protocol analysis tool, Lowe [12] identified a protocol-interaction attack on the 1978 *Needham-Schroeder (NS)* public-key authentication protocol [13], based on a subtle structural flaw which had gone unnoticed for 17 years. We point out a similar attack on SecureDNA's basic request protocol that circumvents rate limiting. We first discovered this flaw through our formal-methods analysis.

Aside from its monitoring, SecureDNA depends crucially on four mechanisms to achieve its security goals: a *custom mutual-authentication protocol (SCEP)*, *exemption-list tokens (ELTs)* to implement its exemption-list feature, an associated *certificate management system*, and *Transport Layer Security (TLS)* [14], [15], [16] to protect communication channels. SecureDNA uses TLS 1.3 and 1.2, but only with ECDH and not with RSA. If TLS fails, then a *Man-in-the-middle (MitM)* attack would be possible and the only security goal that SecureDNA would achieve would be secrecy of orders.

Our analysis uncovers undesirable vulnerabilities concerning how SecureDNA uses these mechanisms, violating the principle of defense in depth. First, due to a structural weakness, SecureDNA's custom authentication protocol provides only one-way authentication: the hazards database and keyservers never learn with whom they communicate. Consequently, an adversary, who tricks an honest synthesizer $\mathcal{S}$ into connecting with a malicious or corrupted database or keyserver, could masquerade as $\mathcal{S}$ to circumvent rate limiting. Our proof-of-concept implementation of this MitM attack demonstrates its feasibility.

Second, structural protocol weaknesses involving inadequate cryptographic bindings of ELTs, authentication cookies, and responses from the hazards database, prevent the system from detecting if responses, within a TLS channel, from the hazards database were modified. These inadequate bindings create a "latent vulnerability": if a synthesizer were to reestablish a connection with the database over the same TLS session, an adversary could replay and swap responses from the database without breaking TLS. Although the current SecureDNA implementation does not allow such reconnections, the vulnerability might manifest under future implementations or TLS configurations.

Recently we made a responsible disclosure of our findings to the SecureDNA team, and we had fruitful discussions with them. We learned that part of their (undocumented) security strategy is based significantly on monitoring, detecting, and responding to certain types of malicious behaviors rather than on preventing such behaviors. They stated that, with the aid of source code that we have not seen, their strategy includes automatic and manual intervention based on automatic alerts, which aim to detect unexpected deviations from typical behaviors, whether due to errors or malice. Their whitepaper does not mention this strategy.

Our contributions include: (1) A validated discernment of the SCEP, basic request, and exemption-handling protocols of the SecureDNA system based in part on an examination of the source code. (2) CPSA models of these protocols. (3) Formal statements of security goals of these protocols, and formal-methods symbolic analysis of the models for structural weaknesses with respect to these goals. (4) Identification of structural flaws in SecureDNA's SCEP protocol, showing that the protocol achieves only one-way authentication. We also exhibit rate-limiting and *denial-of-service (DoS)* attacks based on this vulnerability. (5) Identification of structural weaknesses in the SecureDNA protocols involving inadequate authentication and cryptographic bindings, including of ELTs, authentication cookies, and responses from the hazards database. We also explain how an adversary could exploit this vulnerability to replay and swap responses from the database, without breaking TLS, if a synthesizer were to reconnect with the database using the same TLS session. (6) Recommendations for strengthening the SecureDNA system, and formal-methods validation of our suggested improvements to SCEP and to authentication and the bindings of protocol message components.

Section IX presents our formal-methods analysis of SCEP, and Section X analyzes our improved SCEP+. Appendix C summarizes the results of our analyses in two tables. All of our artifacts are available on GitHub [17], including our proof-of-concept implementation of our MitM attack and complete CPSA input models and output shapes. For more details, see our full paper [18].

II. PREVIOUS WORK

Aside from the abstract cryptographic studies by Baum et al. [5], [6], and a 2022 course project by Langenkamp et al. [19], to our knowledge, our work is the only security analysis of the SecureDNA system. Whereas Baum et al. describe and analyze a cryptographic protocol for computing DOPRFs when some keyservers are malicious, we analyze the security of the SecureDNA system design, engineering, and implementation, with a focus on its query protocols. Baum et al. do not consider rate-limiting or DoS attacks.

Kane and Parker [20] and Hoffman et al. [21] review the landscape in DNA screening. The U.S. Department of Health

and Human Services [22] offers non-binding security considerations, but there are no standards or laws that require screening for dangers. Founded in 2009, the International Gene Synthesis Consortium is a coalition of synthetic DNA providers and stakeholders that use a common screening protocol [23]. Johns Hopkins [24] maintains a hub of useful information, including a list of screening companies and tools. We are not aware of any previous work that analyzes the design, security engineering, and implementation of any such system. Previous work deals mainly with biosafety, not cybersecurity.

III. BACKGROUND

We present brief relevant background regarding protocol analysis, strand spaces, and CPSA. For more details about these topics, see [25].

A. Protocol Analysis

Formal-methods analysis of a protocol involves expressing the protocol in a formal mathematical model, stating propositions that reflect the protocol's desired security properties, and proving or disproving those propositions. Often, this process requires the assistance of specialized theorem-proving tools, such as ProVerif [26], Tamarin Prover [27], Maude-NPA [28], or CPSA [29]. *Symbolic*, versus computational, formal-methods analysis looks for *structural* (fundamental logical) weaknesses, not cryptographic weaknesses. Such formal-methods analysis of protocols will not detect implementation errors nor the application of protocols to inappropriate settings.

B. Strand Spaces

Strand spaces [30] are a useful symbolic formalism for modeling the authentication and secrecy properties of cryptographic protocols. In the strand-space formalism, a cryptographic protocol is a set of *roles* that form a template for legitimate strands. A *strand* is a sequence of sent and received messages, where each message is an element of a term algebra that contains operations such as encryption and message concatenation. The *strand space* for a cryptographic protocol is the set of all strands formed by term substitutions on the roles of the protocol or adversary strands.

In the strand space formalism, executions of a protocol are modeled as *bundles*. A bundle for a protocol $\mathcal{P}$ is a set of strands (or prefixes of strands) from the strand space of $\mathcal{P}$ such that every reception node in the bundle corresponds to a unique transmission node in the bundle that sends the same message that was received. The directed graph with edges connecting consecutive nodes on the same strand, and connecting corresponding reception and transmission nodes, must be acyclic so that the events in the bundle respect causality. Bundles are of central importance in formal-methods analysis using strand spaces because each bundle provides an interpretation of a security goal formula. A protocol achieves a security goal if and only if the security goal is true under the interpretation of all of the protocol's bundles.

C. CPSA

CPSA [29] is an open-source tool for analyzing cryptographic protocols within the strand-space model. CPSA distinguishes itself as a *model-finder*. Its input is a model, which comprises strands consisting of roles, messages, variables, and a set of initial assumptions. When executing to completion, CPSA provably identifies all essentially different executions of the protocol within a *Dolev-Yao (DY)* network intruder model [31] and outputs them as *"shapes"* [32]. CPSA's model finding enables users to identify the strongest achieved security goal for an input model [33]. Users define CPSA models using LISP-like s-expressions that implement a custom language. In these models, which superficially resemble (but are not) executable source code, users specify one or more roles, associated variables and messages, and assumptions.

IV. SECUREDNA SYSTEM

The SecureDNA system [3] provides a way for honest synthesizers to screen DNA synthesis-order requests for biohazards. We summarize how this system works by explaining its architecture, security goals, oblivious search for hazards, certificate infrastructures, authentication tokens, exemption-list tokens, and source code.

A. Architecture

As envisioned by its inventors, the SecureDNA system involves eight logical entities: a *Customer* $\mathcal{C}$ sends an order request, and a *Synthesizer* $\mathcal{S}$ [E10][1] receives and processes the order request. There is a *Plain Text Database* $\mathcal{D}$ of hazards, and a *Keyed Hashed Database* $\mathcal{H}$ [E7] of hazards. A *Curator* $\mathcal{R}$ populates $\mathcal{D}$. A *Biosafety Officer* $\mathcal{B}$ grants exemption tokens to $\mathcal{C}$. Using Shamir secret sharing of a hash key k, a *Distributed Keyserver* $\mathcal{K}$ [E4] applies keys needed to populate and query $\mathcal{H}$. A *SecureDNA Administrator* $\mathcal{F}$—the SecureDNA Foundation—is the singular root of trust that generates the hash key, establishes the root certificate for each of the three certificate hierarchies, processes requests for synthesizer and exemption certificates, and releases system software and updates. In addition, it is prudent to include a ninth entity, an *Authentication Backend* $\mathcal{A}$, that verifies authentication requests for exemption tokens. See Figure 1.

B. Security Goals

Based on the SecureDNA whitepaper [3], we understand that the SecureDNA system aims to accomplish four security goals, which we state informally:

SG1 Keep the sequences in $\mathcal{D}$ secret to everyone except the Curator $\mathcal{R}$.

SG2 Keep each synthesis order request s secret to everyone except $\mathcal{C}$ and $\mathcal{S}$.

SG3 Return to $\mathcal{S}$ a valid answer to whether a synthesis order request s is in $\mathcal{D}$.

SG4 When $\mathcal{C}$ presents an ELT to $\mathcal{S}$, enable $\mathcal{S}$ to determine whether $\mathcal{C}$ is authorized to receive the synthesis of order request s, even if some sequences in s are in $\mathcal{D}$.

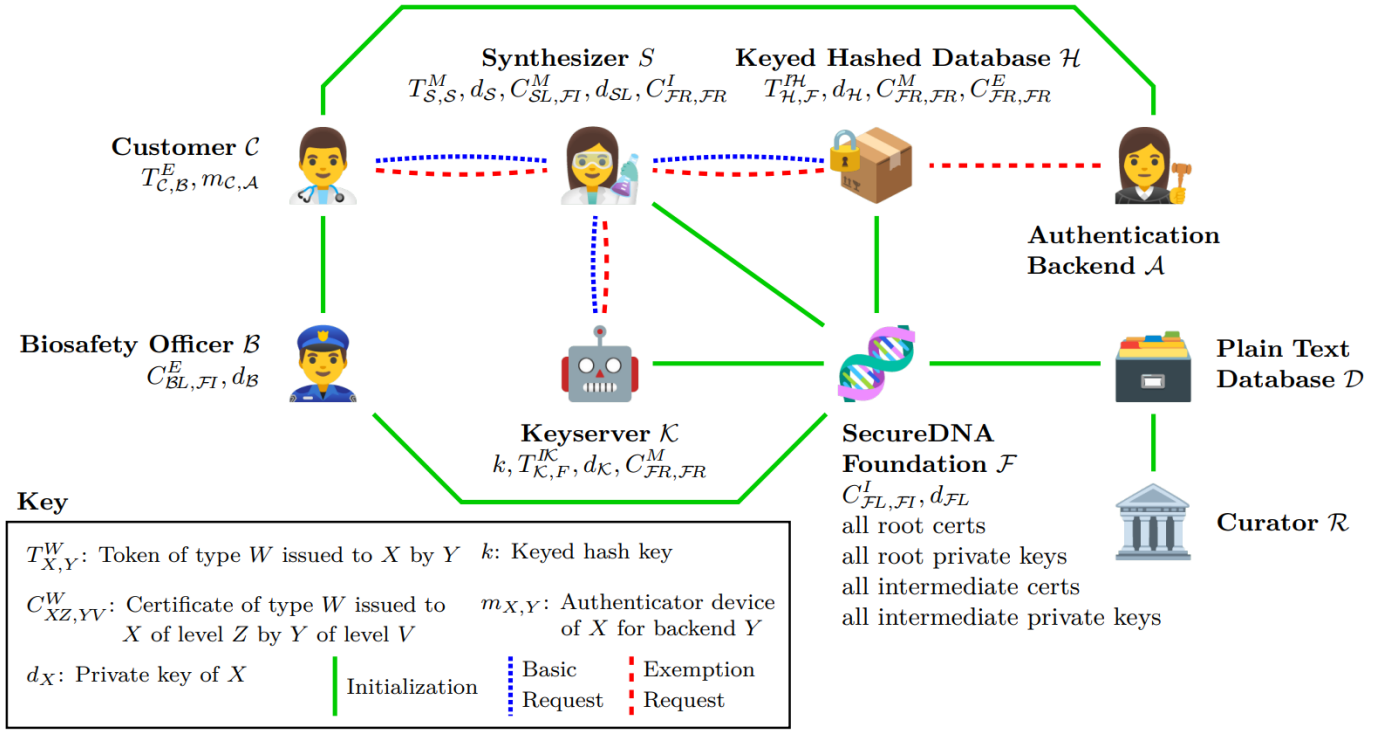[1]The notation "[E10]" means see Endnote 10 in Appendix J of [18].

Fig. 1. Architecture of the SecureDNA system showing the roles and cryptographic variables held by each role.

Neither the SecureDNA whitepaper [3] nor the initial technical cryptographic manuscript [6] states any of these goals precisely. The more recent cryptographic article [5] deals only with the abstract mathematical cryptography and does not address SG3 or SG4. Instead, this article focuses on ensuring correct operations of $\mathcal{K}$ when some of the distributed keyservers are malicious.

To analyze these goals, it is essential to identify the adversarial model (see Section VI). The whitepaper [3] vaguely mentions the *semi-honest* and *malicious* models without clearly articulating what model is applied in what context. The semi-honest model is a very weak model (participants must follow the protocol) and usually is insufficient for meaningful protection in many network environments. Following well-accepted practice, we adopt the DY model.

Concerning SG1, documentation states that "secrecy [of the database of hazards] should be protected at the highest possible level while preserving usability," [6, p. i] and "leakage of $\mathcal{H}$ about $\mathcal{D}$ must be kept to a minimum." [3, p. 17].

However, when we presented our rate-limiting attack to the SecureDNA team, they asserted that SG1 is not very important, in part, because most known hazards are already described in publicly available documents. They explained that one important aspect of SG1 concerns the relatively few newly discovered hazards that are not yet widely known. We consider SG1 an important goal; much simpler designs would be possible if SG1 were eliminated.

Another important aspect of SG1 involves SecureDNA's subtle biological strategy for detecting functional mutants of

hazards, which are too numerous to list individually in $\mathcal{D}$. Every "*re-spinning*" of $\mathcal{H}$ involves selecting new representative mutants to include in $\mathcal{D}$, then *rehashing* $\mathcal{H}$. To evade detection of mutants, it would be helpful to know which mutants are in $\mathcal{D}$. An adversary might attempt a *database-scraping attack* to learn parts of $\mathcal{D}$ and to learn which mutants are in $\mathcal{D}$. Re-spinning complicates learning what mutants are in $\mathcal{D}$ by changing them, but re-spinning does not prevent attacks that aim to learn the non-mutant entries of $\mathcal{D}$. Rehashing only makes such attacks less efficient. Although monitoring and re-spinning provide significant defenses, better protocols would provide stronger protections.

We sensed that SecureDNA is extremely concerned about adoption, and to that end, their most important security goal is SG2. They are also very concerned about non-security goals, including speed and quality of service.

C. Screening for Hazards

The SecureDNA system screens for biohazards by performing a type of oblivious search that uses a keyed hashed database $\mathcal{H}$ of hazards. Because of its dangerous information, and to complicate evading detection of hazardous variants, the corresponding plain text database $\mathcal{D}$ must be kept secret. Sequences in $\mathcal{D}$ are relatively short (as small as 60 bits) to prevent the synthesis of longer hazards from shorter sequences.

The system depends in part on rate-limiting queries to $\mathcal{H}$ to prevent an adversary from searching a large number of the possible short sequences, even if an exhaustive search of such sequences were computationally infeasible. The system also

periodically re-spins $\mathcal{H}$ to thwart the adversary from learning which mutants are in $\mathcal{H}$ (see Section IV-B).

D. Certificate Infrastructures

The SecureDNA system manages a custom certificate-based *public-key infrastructure (PKI)* to support authentication and exemptions. There are three separate hierarchies: *manufacturer* (to authenticate $\mathcal{S}$), *infrastructure* (to authenticate $\mathcal{K}$ and $\mathcal{H}$), and *exemption* (to prove $\mathcal{C}$ is authorized to synthesize some exempted sequences). Each hierarchy has a separate root of trust created by the Administrator $\mathcal{F}$: *manufacturer root* $C^M_{\mathcal{FR},\mathcal{FR}}$, *infrastructure root* $C^I_{\mathcal{FR},\mathcal{FR}}$, and *exemption root* $C^E_{\mathcal{FR},\mathcal{FR}}$. SecureDNA distributes these roots of trust in its software release.

Each node in the hierarchy is a *certificate*, which cryptographically binds an identity with its public key. Each non-root certificate is digitally signed by the private key of its parent in the hierarchy. Each certificate has the following format:

$$C^W_{x,y} = (N_x, context, \{M(N_x, context)\}_{d_y}), \quad (1)$$

where

$$context = h, \sigma, p_x, N_y, p_y, \delta. \quad (2)$$

x is the subject (receiver); y is the issuer; M is a cryptographic hash function; and d_y is the private key of y. For any data v and any key z, $\{v\}_z$ denotes encryption or signature of v under key z. Here, h is a description of the certificate, which contains a version, type W (manufacturer, infrastructure, exemption), and hierarchy level (root, intermediate, leaf). σ is a randomly generated identifier assigned to the certificate, which identifies x. N_x and N_y are the identifiers of x and y, respectively, containing a name and email. p_x is the public key of x; p_y is the public key of y; and δ is a validity period, comprising a start and end date [E1,E2,E3]. In our notation $C^W_{XZ,YV}$, W is the type from h; X is the subject; Z is X's hierarchy level; Y is the issuer; and V is the issuer's hierarchy level.

There are special types of certificates, called *tokens*, including *authentication tokens* and *exemption-list tokens*. The format of each token is similar to that of a certificate:

$$T^\pi_{x,y} = (\pi, u_\pi, context, \{M(\pi, u_\pi, context)\}_{d_y}), \quad (3)$$

where π is a certificate type, and the contents of field u_π depend on the certificate type π.

Each token is a leaf of a certificate chain. The usage of "*leaf*" in the hierarchy level of certificates is a misnomer, as leaf certificates are intermediate certificates used to issue tokens. We will use SecureDNA's notation throughout the paper. Each token is bundled with a chain of certificates that can be verified to the corresponding root. SecureDNA manages the creation and distribution of root certificates, creation of intermediate certificates, usage of intermediate certificates to sign leaf certificates, and revocation of certificates. Our notation for tokens is similar to that for certificates, but with no hierarchy level for subjects or issuers, because tokens have no hierarchy, and all issuers are leaf certificates.

$\mathcal{F}$ provides $\mathcal{K}$ and $\mathcal{H}$ with a list of token identifiers σ and public keys p_x for revoked tokens. The SecureDNA team explained that they did not incorporate standard revocation protocols for X.509 (such as OSCP [34]) because of the additional burden on SecureDNA. Similarly, they did not empower $\mathcal{S}$ to act on revocation lists for $\mathcal{K}$ and $\mathcal{H}$. Instead, if $\mathcal{K}$ or $\mathcal{H}$ is compromised, they plan to re-key $\mathcal{H}$. Whenever the root key is changed, SecureDNA would also have to issue a new client, because the root certificates are embedded into the client software.

E. Authentication Tokens

SecureDNA uses three types of authentication tokens: $\mathcal{S}$ uses *Synthesizer tokens* $T^M_{x,y}$ to authenticate to $\mathcal{H}$ and $\mathcal{K}$ [E11,E12]. Each has a u_M field containing an identifier for the synthesizer, and a rate limit μ for the synthesizer. These tokens are bundled with chains rooted in $C^M_{\mathcal{FR},\mathcal{FR}}$. Keyserver $\mathcal{K}$ uses *Keyserver infrastructure tokens* $T^{I\mathcal{K}}_{x,y}$ to authenticate to $\mathcal{S}$. Each has a $u_{I\mathcal{K}}$ field containing $\mathcal{K}$'s ID in the secret sharing scheme [E5,E6]. $\mathcal{H}$ uses *Database infrastructure* tokens $T^{I\mathcal{H}}_{x,y}$ to authenticate to $\mathcal{S}$. Each has an empty $u_{I\mathcal{H}}$ [E8,E9]. The keyserver and database infrastructure tokens are bundled with chains rooted in $C^I_{\mathcal{FR},\mathcal{FR}}$.

F. Exemption-List Tokens

$\mathcal{C}$ uses *exemption tokens* $T^E_{x,y}$ to permit synthesis of dangerous sequences. Each has a u_E containing a list of exempt sequences s_E, and an identifier $m_{\mathcal{C},\mathcal{A}}$ for an *authenticator device* (e.g., YubiKey) issued by $\mathcal{A}$ [E13,E14]. These tokens are bundled with chains rooted in $C^E_{\mathcal{FR},\mathcal{FR}}$. The public key p_y in $T^E_{x,y}$ is optional, and is not used for authenticating token users. The only use of the key pair (p_y, d_y) is to issue sub-tokens $T^{E'}_{x,y}$, which are copies of $T^E_{x,y}$ with the restriction that the sequences in $u_{E'}$ must be a subset of the sequences in u_E.

G. Source Code

The SecureDNA system is written mostly in the Rust programming language [35], which enforces memory safety. We analyzed Version 1.0.8 of the source code [9], which comprises approximately 64,000 lines of source code across nearly 300 files. SecureDNA adopts a TLS implementation from a Rust package. To our knowledge there is no external documentation. There are only sparse comments for the internals of the code and only sparse descriptions of user-facing functionality. $\mathcal{F}$ distributes software using a trusted Linux package manager.

V. SecureDNA Protocols

We explain the two main protocols that the Synthesizer $\mathcal{S}$ uses to screen order requests for hazards: the basic order-request protocol and the exemption-handling protocol. These protocols transform sequences s using a keyed DOPRF $f_k(s) = M(s)^k$, where k is a key and M is a cryptographic hash function. These computations take place in a prime-order group in which the Decisional *Diffie-Hellman (DH)* Problem is hard [36]. Figures 2–3 summarize how these protocols work; for more details, see the associated Ladder Diagrams 4–5.
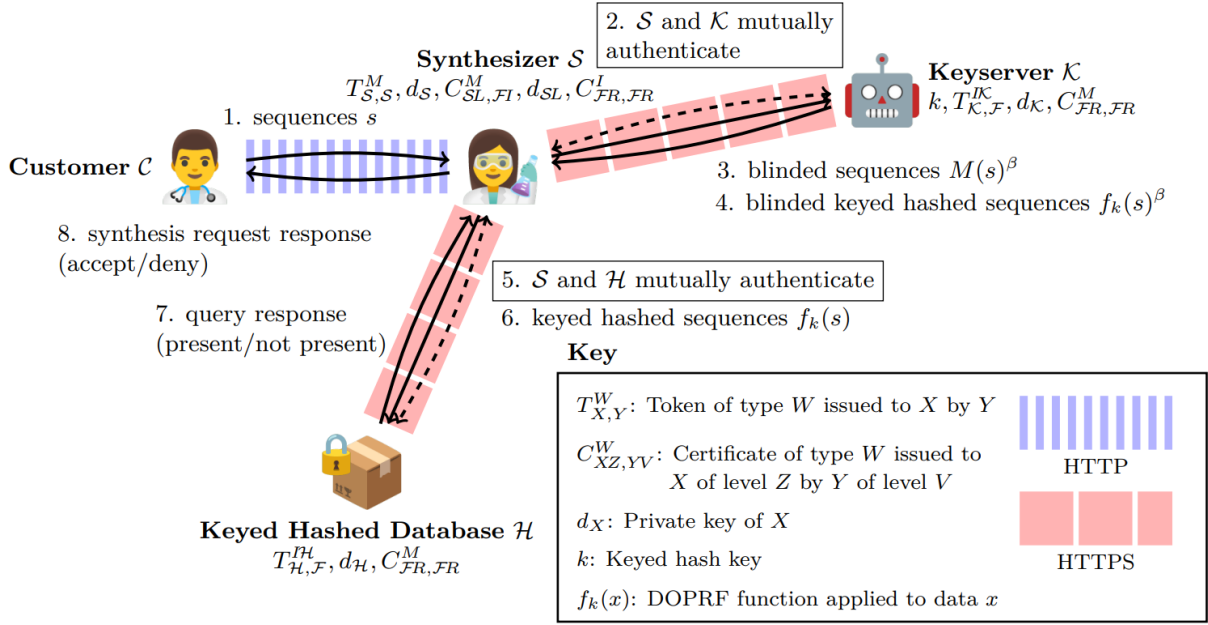
Fig. 2. The basic order-request protocol. Steps 2 and 5 call a custom mutual authentication protocol SCEP (see Section VIII-A).

When $\mathcal{S}$ communicates with the Keyserver or the Hashed Database $\mathcal{H}$, the entities first establish authentication using a custom authentication sub-protocol SCEP that involves exchanging nonces and authentication tokens (see Figures 4–5).

A. Basic Order-Request Protocol

As shown in Figure 2, the basic order-request protocol begins with the Customer $\mathcal{C}$ sending its synthesis request s to $\mathcal{S}$. Synthesizer $\mathcal{S}$ blinds s and sends it to $\mathcal{K}$ (actually, to several keyservers). The keyservers apply their keyshares, and return the results to $\mathcal{S}$. Then $\mathcal{S}$ combines the responses from the keyservers and unblinds the result to form the keyed hash of the request. $\mathcal{S}$ sends this keyed hash to $\mathcal{H}$. Next, $\mathcal{H}$ responds stating whether the keyed hash is in its database. Finally, $\mathcal{S}$ reports back to $\mathcal{C}$ whether their synthesis request is allowed or denied. Because the communication between $\mathcal{C}$ and $\mathcal{S}$ is intended to be hosted locally, by default this communication occurs via HTTP; optionally, HTTPS can be used. Communication between $\mathcal{S}$ and $\mathcal{H}$, and between $\mathcal{S}$ and $\mathcal{K}$, occurs in a one-way authenticated TLS channel supported with certificates for $\mathcal{H}$ and $\mathcal{K}$. At the beginning of these communications between $\mathcal{S}$ and $\mathcal{H}$, and between $\mathcal{S}$ and $\mathcal{K}$, the roles complete a custom mutual authentication protocol SCEP (see Section VIII-A).

B. Exemption-Handling Protocol

As shown in Figure 3, the exemption protocol is similar to the basic order-request protocol. C sends to $\mathcal{S}$ its synthesis request s, exemption-list token $T^E_{\mathcal{C},\mathcal{B}}$ obtained from the Biosafety Officer $\mathcal{B}$, and authenticator code a (a one-time passcode) from the authentication device listed in the exemption-list token. $\mathcal{S}$ performs the same exchange involving $\mathcal{S}$ with $\mathcal{K}$ as in the basic order-request protocol. Then, $\mathcal{S}$ performs a second round with $\mathcal{K}$ to hash each of the sequences in $T^E_{\mathcal{C},\mathcal{B}}$. After $\mathcal{S}$ has assembled all keyed hashed sequences, $\mathcal{S}$ sends to $\mathcal{H}$ the keyed hash sequences, $T^E_{\mathcal{C},\mathcal{B}}$, and a. Then, $\mathcal{H}$ verifies the appropriateness of $T^E_{\mathcal{C},\mathcal{B}}$ by sending to the Authentication Backend $\mathcal{A}$ the authenticator device name, a, and a timestamp. $\mathcal{A}$ responds with an OK or error. If $\mathcal{A}$ responds OK, $\mathcal{H}$ checks if the keyed hash of s is in its database. If it is, $\mathcal{H}$ checks if s is in the exemption list. $\mathcal{H}$ then reports back to $\mathcal{S}$ if the keyed hash request is in its database, and if so, if it is in the exemption token. $\mathcal{S}$ reports back to $\mathcal{C}$ whether their synthesis request is granted. The communication from $\mathcal{H}$ to $\mathcal{A}$ is an HTTPS request. Communication between $\mathcal{S}$ and $\mathcal{H}$, and between $\mathcal{S}$ and $\mathcal{K}$, occurs in a one-way authenticated TLS channel supported with certificates for $\mathcal{H}$ and $\mathcal{K}$. At the beginning of these communications between $\mathcal{S}$ and $\mathcal{H}$, and between $\mathcal{S}$ and $\mathcal{K}$, the roles complete a custom mutual authentication protocol SCEP (see Section VIII-A).

VI. ADVERSARIAL MODEL

We assume a malicious model in which a protocol communicant does not have to follow the protocol. More specifically, we assume the DY model in which the adversary has full control over all messages on the network and can manipulate an unbounded number of protocol sessions. The adversary can perform the roles of the legitimate communicants. The adversary cannot break cryptographic primitives but can compute keyed cryptographic primitives when the adversary knows the keys. Objectives of the adversary include learning the sequences in the order request, learning some or all plain text entries in the hazards database, and causing the synthesizer to synthesize a sequence in the hazards database (for which the adversary does not have permission to synthesize).
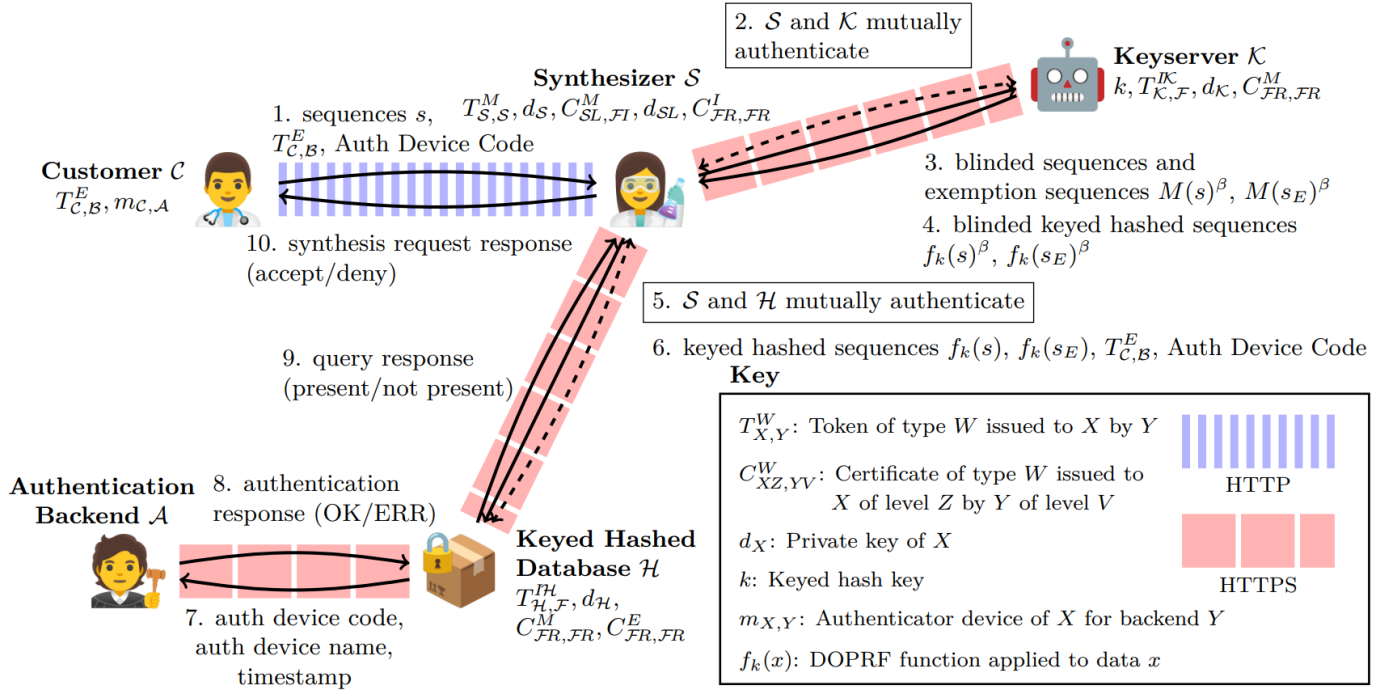
Fig. 3. The exemption-handling protocol. $\mathcal{C}$'s request includes an ELT signed by $\mathcal{B}$ and a one-time passcode (the "Auth Device Code"). In $\mathcal{H}$'s response to $\mathcal{S}$, $\mathcal{H}$ sends a list of sequences that are in its database, with boolean flags indicating which sequences are also in the exemption token. $\mathcal{H}$ verifies that $\mathcal{C}$ included a valid passcode. Steps 2 and 5 call a custom mutual authentication protocol SCEP (see Section VIII-A).

Inherent limitations of the system include that the customer and synthesizer learn whether the order request is in the Database $\mathcal{D}$, and a malicious synthesizer can synthesize any sequence.

VII. CIRCUMVENTING RATE LIMITING

SecureDNA includes *rate-limiting mechanisms* that aim to make exhaustive "dictionary attacks [on $\mathcal{D}$] impossible" by limiting the number of queries that can be made on $\mathcal{H}$ in a given time [3, p. 3]. We explain how rate limiting works and point out an attack that exploits the failure of SCEP to provide mutual authentication.

The main stated purpose of rate limiting is to achieve SG1. The SecureDNA team explained that another purpose is to detect software errors. We note that rate limiting can also try to defend against certain types of DoS attacks. Without debating the significance of SG1 (see Section IV-B) and without discussing technical biological issues involving the relationship of SG1 to determining functional variants of hazards, we will analyze SecureDNA's rate-limiting mechanisms.

A. How Rate Limiting Works

SecureDNA includes two classes of rate-limiting mechanisms: (1) software mechanisms that attempt to limit the number of times $\mathcal{K}$ executes the DOPRF, which limits the number of queries $\mathcal{S}$ can make on $\mathcal{H}$, and (2) auditing mechanisms that attempt to monitor, detect, and react to unreasonable behaviors, implemented by non-public software (which we have not seen) and human intervention.

The software mechanisms work as follows. Every time $\mathcal{S}$ executes the basic-request or exemption-handling protocol, $\mathcal{K}$ and $\mathcal{H}$ separately record in their own query-databases the time and number of sequences requested by $\mathcal{S}$. Entities $\mathcal{K}$ and $\mathcal{H}$ index their query-database by the identifier σ in $\mathcal{S}$'s authentication token $T_{\mathcal{S},\mathcal{S}}^M$. Entities $\mathcal{K}$ and $\mathcal{H}$ check if the total number of sequences requested by $\mathcal{S}$ within the previous 24 hours exceeds the rate limit μ in $T_{\mathcal{S},\mathcal{S}}^M$ [E20].

The SecureDNA team explained that auditing mechanisms provide their main defense against dictionary attacks and certain other malicious behaviors. They stated that their reactive capabilities include the ability to revoke tokens, certificates, and entire certificate chains.

B. Potential Vulnerabilities

The failure of SCEP to provide mutual authentication is a serious vulnerability for rate limiting and DoS: because neither $\mathcal{H}$ nor $\mathcal{K}$ know with whom they are communicating, it is difficult to identify which actor is initiating a request. The SecureDNA team stated that knowing the identity of the initiator is not necessary because queries are associated with authentication tokens, not actors. We will show, however, that it is possible for the adversary to steal and misuse authentication tokens.

Concerning the software mechanisms, the following facts might in some contexts be potential vulnerabilities [E21]: (1) $\mathcal{F}$ has no control over the rate limit μ: $\mathcal{S}$ creates its own authentication token $T_{\mathcal{S},\mathcal{S}}^M$, setting its own μ, which can be any 64-bit integer. Note, however, that servers now limit queries

operationally (see Appendix B). (2) $\mathcal{F}$ has no control over the token identifier σ in $T_{\mathcal{S},\mathcal{S}}^{M}$, which $\mathcal{S}$ chooses. (3) There is no limit on the number of authentication tokens $\mathcal{S}$ can create. (4) $\mathcal{S}$ can manufacture any number of valid leaf certificates. (5) The Auth Device Code is not bound to context, enabling a corrupt $\mathcal{K}$ or $\mathcal{H}$ to steal and misuse it. The SecureDNA team explained that they do not view these facts as vulnerabilities because they guard against rate-limiting attacks primarily with auditing and re-spinning. They also explained that, to avoid undue administrative burden, they do not want $\mathcal{F}$ to authorize tokens $T_{\mathcal{S},\mathcal{S}}^{M}$ created by $\mathcal{S}$.

C. Attacks

(1) We describe a rate-limiting attack that exploits the weakness that SCEP provides only one-way authentication. In this attack, the adversary is a malicious $\mathcal{K}'$, which is possible within SecureDNA's adversarial model [5], [6] and our DY model. For example, $\mathcal{K}'$ might be the adversary performing the role of $\mathcal{K}$ or it might be a compromised $\mathcal{K}$. An honest $\mathcal{S}$ connects with $\mathcal{K}'$, which enables $\mathcal{K}'$ to learn $\mathcal{S}$'s authentication token without breaking TLS. This connection is possible if $\mathcal{S}$ trusts a compromised signing key included in $\mathcal{K}'$'s certificate chain. Then, following a strategy similar to that in Lowe's [12] attack on NS, $\mathcal{K}'$ connects with an honest $\mathcal{K}$ masquerading as $\mathcal{S}$, using $\mathcal{S}$'s authentication token. After authenticating, the adversary $\mathcal{K}'$ now can issue as many queries as they desire without detection, up to the rate limit for $\mathcal{S}$ and the auditing limit. From the perspectives of $\mathcal{K}$ and auditors, the queries are associated with $\mathcal{S}$. See Appendix D for details.

The adversary $\mathcal{K}'$ can repeat this attack exploiting different synthesizers without detection by the software controls. Eventually, the auditing controls would likely notice a suspicious increase in the total volume of requests, even though they would not necessarily know the cause of this increase. The SecureDNA team stated that their monitoring software would identify the offending server.

As a proof-of-concept, we implemented our MitM attack against a corrupt keyserver. The attack works as we envisioned. For source code and pcap files, see our artifacts [17].

A very similar rate-limiting attack is also possible involving a corrupt $\mathcal{H}'$, who can masquerade as $\mathcal{S}$ to $\mathcal{H}$. The attack can also be adapted as a DOS attack, using ideas from (2).

(2) Before we learned of SecureDNA's auditing strategy, we pointed out that, if the system worked solely as defined in the publicly available software, then an adversary could carry out certain malicious activities without detection. For example, due to intentional design and implementation features, anyone could circumvent query rate limits by setting a high rate limit or generating many additional nodes and tokens in the certificate hierarchy. Also, an adversary could cause DoS by creating and misusing tokens with identifiers that collided with those of legitimate tokens, prompting SecureDNA to revoke legitimate tokens (e.g., for exceeding rate limits).

We observe, however, that the adversary could not in other contexts directly reuse a harvested authentication token because the adversary would also need its private key. The adversary might not be able to reuse an ELT directly because the ELT is also protected by a two-factor device m.

D. Risks

The risk of Attack (1) is low. The adversary must trick synthesizers into connecting with $\mathcal{K}'$, for example, by compromising $\mathcal{K}'$ or manipulating the management of root certificates. Eventually auditing will detect a suspicious increase in total volume of queries, including queries associated with $\mathcal{K}'$ or $\mathcal{H}'$; and there is limited value in learning $\mathcal{D}$. Also, a malicious $\mathcal{K}'$ or $\mathcal{H}'$ could carry out other simpler attacks, such as disabling rate limiting or returning incorrect answers. Nevertheless, the adversary would likely be able to learn some entries of $\mathcal{D}$. It would be stronger security engineering to prevent this protocol-interaction attack by requiring mutual authentication, as Release 1.1.0 now does.

The SecureDNA team stated that Attacks (2) would be detected and shut down by their auditing mechanisms, and that collisions would be detected upon revocation.

VIII. WEAK AUTHENTICATION AND INADEQUATE BINDINGS

We identify structural weaknesses in SecureDNA's SCEP custom mutual authentication protocol and in bindings of ELTs, authentication tokens, and responses from $\mathcal{H}$. These vulnerabilities violate the principle of defense in depth, and the binding weakness permits a response-swapping attack if $\mathcal{S}$ were to reconnect with $\mathcal{H}$ over the same TLS session (which the implementation disallows). We begin by explaining SCEP.

A. Server Connection Establishment Protocol (SCEP)

As shown in Figure 4, in the basic-request and exemption-handling protocols, when $\mathcal{S}$ communicates with $\mathcal{K}$ or $\mathcal{H}$, rather than using *mutual TLS (mTLS)* [37], SecureDNA engages in a custom mutual authentication protocol, called the *Server Connection Establishment Protocol (SCEP)* [E15,E16,E17]. For a discussion of this design choice, see Section XI. SecureDNA's documentation clearly reveals that SecureDNA intended for SCEP to achieve mutual authentication—for example, the file containing the functionality is called "mutual_authentication.rs" [E22]. SCEP begins by establishing a TLS channel, which by default is based on one-way authentication. During SCEP, $\mathcal{S}$ receives a *HTTPS request cookie* ω, which $\mathcal{S}$ uses to authenticate future messages in the basic-request or exemption-handling protocol. This cookie ω is different from a SecureDNA authentication token.

B. Vulnerabilities

As shown in Section IX, SCEP does not achieve mutual authentication. $\mathcal{S}$ authenticates the server ($\mathcal{H}$ or $\mathcal{K}$), but the server does not know with whom it is communicating. The security properties of TLS with SCEP, including with respect to authentication, are only slightly greater than those of TLS alone. In terms of Lowe's authentication hierarchy [38], using SCEP adds only the lowest guarantee—*liveness*. This vulnerability stems from the omission of session parameters

ω and the server certificate from $\mathcal{S}$'s signed response; that is, the vulnerability stems from a failure to cryptographically bind the response to the context.

Similarly, in the basic-request and exemption-handling protocols, neither any response from $\mathcal{K}$ nor any response from $\mathcal{H}$ is cryptographically bound to $\mathcal{S}$'s request. Consequently, an adversary might—in some situations—be able to replay and swap these responses out of context, as we will next show. The one-time passcode a sent by $\mathcal{C}$ is not bound to the context and hence can be reused by any malicious recipient.

C. A Latent Response-Swapping Attack

SecureDNA's weak authentication and binding are undesirable characteristics that violate the principle of defense in depth. Section VII explains a rate-limiting attack made possible by SCEP's one-way authentication.

We now explain how the inadequate binding of $\mathcal{H}$'s responses might permit a latent response-swapping attack without violating the TLS channel. Suppose $\mathcal{S}$ *reconnects* with $\mathcal{H}$ over the same TLS session (with the same keys). The adversary could then potentially violate SG3 and SG4 by swapping a query response in the second connection with the corresponding one from the first connection. This hypothetical attack does not assume a malicious $\mathcal{S}$, $\mathcal{H}$, or $\mathcal{K}$.

To carry out this attack without detection, the adversary would have to deal with the following technical issues, which might depend in part on the TLS implementation. First, the response swapping can be accomplished by manipulating bits at the TCP network layer, which is easy to do [39]. Second, the swapping must not change the TLS MAC value [15], [16], which depends in part on the sequence number. For TLS 1.2 and 1.3, the sequence number is direction specific and always begins with zero [15], [16]. Thus, responses in the same locations from the two connections using the same keys will have the same sequence numbers and thus can be swapped without changing the MAC value.

We observe, however, that if the adversary modified messages sent from $\mathcal{S}$ to $\mathcal{K}$, the adversary would be detected given the "active security" measures in place for protecting against malicious $\mathcal{K}$'s [5].

D. Risks

The risk of the response-swapping attack depends in part on the likelihood of $\mathcal{S}$ reconnecting with $\mathcal{H}$ over the same TLS session. We understand that the SecureDNA implementation does not permit reconnections or 0-RTTs, so this attack is not possible. Specifically, by default, TLS 1.3 implementations set reconnections and 0-RTT off [E23]. If, however, a future SecureDNA implementation allowed reconnections, the consequence could be that $\mathcal{S}$ takes incorrect actions, synthesizing hazards and denying legitimate synthesis requests. Although the overall risk might be relatively low, it would be stronger security engineering to avoid this latent vulnerability.

The new Version 1.1.0 release includes an optional mode (which we have not studied)—called *verifiable screening*— in which $\mathcal{S}$ sends a hash of the query to $\mathcal{H}$, and $\mathcal{H}$ returns to $\mathcal{S}$ a signed hash of several items including the hash of the query, $\mathcal{H}$'s response, and metadata. Although the purpose is to enable $\mathcal{S}$ to save proof that it had screened the order, properly executed this strategy can also solve the binding weakness, albeit inefficiently.

IX. FORMAL-METHODS ANALYSIS OF SCEP

We apply the strand-space formalism to analyze security properties of the custom SCEP protocol (see Section VIII-A), which executes within a one-way authenticated TLS session. The purpose of SCEP is to mutually authenticate the synthesizer $\mathcal{S}$ with the infrastructure system $\mathcal{K}$ or $\mathcal{H}$. To carry out our analysis, we (1) define a *strand space* (Definition IX.1) for the combined TLS-SCEP protocol, (2) formalize the SCEP mutual authentication security goal as a pair of logical formulas describing properties of distinct execution models on the SCEP strand space, and (3) use CPSA to verify the formal security goals. Our CPSA inputs and outputs are available on GitHub [17].

It is useful to introduce the concept of a *trace*, which is a finite, non-empty sequence of events (message transmissions or receptions) that take place within a designated channel.

Definition IX.1 (Strand Space). A *directed term* is a pair (d, t), where $t \in A$ (A is the set of all possible protocol terms, or messages) and d is either incoming $(-)$ or outgoing $(+)$. A *strand* s comprises a trace of directed terms, where s specifies the actions of a legitimate party or of the adversary. A *strand space* is a set Σ of strands.

A. SCEP Strand Space

To define the SCEP strand space (Definition IX.8), we define *penetrator strands* that model adversarial behavior and *regular strands* that model behavior of legitimate protocol roles. Definition IX.2 specifies the set Σ_{pen} of penetrator strands. Definition IX.7 specifies sets of regular strands that carry out the roles of the synthesizer $\Sigma_{SCEP-\mathcal{SR}}$ and the responder $\Sigma_{SCEP-\mathcal{RS}}$. We use the SCEP strand space, which combines the penetrator and regular strands, to model protocol executions that potentially incorporate adversarial behavior.

Definition IX.2 (Penetrator Strands). Let $T \subseteq A$ be a set of atoms that represent principals, string literals, keys, and nonces. Let $K \subseteq T$ be a set of all encryption keys, including inverse keys. Let $e(t, k)$ be the encryption of term t under key k. Depending on whether k is a symmetric or asymmetric key, $e(t, k)$ may denote either symmetric or asymmetric encryption; k is a symmetric key when $k^{-1} = k$. The set Σ_{pen} is a set of all strands p, where p has one of the penetrator traces below:
1) $Generate(t)$: $[+t]$ where $t \in T$.
2) $Encrypt(g, k)$: $[-g, -k, +e(g, k)]$ where $g \in A, k \in K$.
3) $Decrypt(e(g, k), k^{-1})$: $[-e(g, k), -k^{-1}, +g]$ where $g \in A, k^{-1} \in K$.
4) $Hash(g)$: $[-g, +h(g)]$ where $g \in A$.
5) $Concatenate(g, h)$: $[-g, -h, +g||h]$ where $g, h \in A$.
6) $Separate(g||h)$: $[-g||h, +g, +h]$ where $g, h \in A$.

Because SCEP depends on an existing one-way authenticated TLS connection, we provide and incorporate traces for TLS 1.2 with ephemeral DH key-exchange (Definition IX.4). SCEP supports TLS 1.2 and TLS 1.3 channels, specifying that TLS 1.2 must use an ephemeral DH ciphersuite. Based on a previous analysis [40], traces of TLS 1.2 with ephemeral DH and TLS 1.3 are cryptographically equivalent. SCEP and the underlying TLS connection depend on asymmetric cryptography. In Definition IX.3, the function pk expresses a principal's public key, and sk expresses the corresponding private key.

Definition IX.3 (Public and Private Keys). Let $pk : Name \rightarrow K$ and $sk : Name \rightarrow K$ be one-to-one functions with disjoint images such that $pk(a) = sk(a)^{-1}$ for all $a \in Name$.

Definition IX.4 (TLS Traces). We define traces that establish a TLS handshake between a client C and a server S. First, we define the terms we use within the trace:

- $C, S, CA \in Name$
- $r_C, r_S \in Nonce$
- $e_S, e_C \in Expt$
- $Cert_S : S||pk(S)||e(h(s, g^{es}), sk(CA)))$
- $S_{KE} : g^{es}||e(h(r_C||r_S||g^{es}), sk(S))$
- $PMS : g^{es \times ec}$
- $C_{WRITE} : h(PMS, r_C, r_S, \text{"client-write"})$
- $S_{WRITE} : h(PMS, r_C, r_S, \text{"server-write"})$
- $C_{FMESG} : r_C||r_S||Cert_S||S_{KE}$
- $C_{FIN} :$
  $e(h(PMS||\text{"client-fin"}||C_{FMESG}), C_{WRITE})$
- $S_{FIN} :$
  $e(h(PMS||\text{"server-fin"}||C_{FMESG}||C_{FIN}), S_{WRITE})$.

Let trace
$Tr_{TLS-C}(C, S, CA, r_C, r_S, e_S, e_C) =$

1) $+r_C$
2) $-r_S||Cert_S||S_{KE}$
3) $+g^{ec}||C_{FIN}$
4) $-S_{FIN}$.

Let the complementary trace
$Tr_{TLS-S}(C, S, CA, r_C, r_S, e_S, e_C) =$

1) $-r_C$
2) $+r_S||Cert_S||S_{KE}$
3) $-g^{ec}||C_{FIN}$
4) $+S_{FIN}$.

For the SCEP strand space, we specify a simple token (Definition IX.5) that acts as a certificate of a principal's public asymmetric key. By contrast, the system implementation tokens embed custom SecureDNA certificates and corresponding signatures. From our analysis, this simplification provides equivalent security guarantees (under the assumption that the SecureDNA foundation is an honest *certificate authority (CA)*), while simplifying execution models resulting from the SCEP strand space.

Definition IX.5 (SCEP Token). Let $X, Y \in Name$, $data \in Text$.

Let $Tok_{X,Y} = \text{"Token"}||X||pk(X)||Y||pk(Y)||data$.
We define an SCEP *token* as a compound term with the structure
$T_{X,Y} = Tok_{X,Y}||e(h(Tok_{X,Y}), sk(Y))$.

Definition IX.6 (SCEP Traces). We define traces that carry out SCEP between $\mathcal{S}$ and a responding infrastructure server $\mathcal{K}$ or $\mathcal{H}$, which we define as $\mathcal{W}$. As before, we first define the terms that we use within the traces:

- $\mathcal{S}, \mathcal{W}, M, CA \in Name$
- $r_{\mathcal{S}}, r_{\mathcal{W}}, r'_{\mathcal{S}}, r'_{\mathcal{W}}, \omega \in Nonce$
- $e_{\mathcal{S}}, e_{\mathcal{W}} \in Expt$
- $T_{S,M}$ (Definition IX.5)
- $T_{\mathcal{W},CA}$ (Definition IX.5)
- $\mathcal{S}_{WRITE} : h(g^{es \times ew}, r'_{\mathcal{S}}, r'_{\mathcal{W}}, \text{"client-write"})$
- $\mathcal{W}_{WRITE} : h(g^{es \times ew}, r'_{\mathcal{S}}, r'_{\mathcal{W}}, \text{"server-write"})$.

Let $Tr_{SCEP-\mathcal{SW}}(\mathcal{S}, \mathcal{W}, M, CA, r_{\mathcal{S}}, r_{\mathcal{W}}, r'_{\mathcal{S}}, r'_{\mathcal{W}}, \omega, e_{\mathcal{S}}, e_{\mathcal{W}}) :$

1) $Tr_{TLS-C}(\mathcal{S}, \mathcal{W}, CA, r'_{\mathcal{S}}, r'_{\mathcal{W}}, e_{\mathcal{S}}, e_{\mathcal{W}})$ (Definition IX.4)
2) $+e(r_{\mathcal{S}}||T_{S,M}, \mathcal{S}_{WRITE})$
3) $-e(\omega||r_{\mathcal{W}}||T_{\mathcal{W},CA}$
   $||e(h(\text{"server-mutauth"}, r_{\mathcal{S}}, r_{\mathcal{W}}, T_{\mathcal{W},CA}), pk(\mathcal{W})),$
   $\mathcal{W}_{WRITE})$
4) $+e(\omega$
   $||e(h(\text{"client-mutauth"}, r_{\mathcal{S}}, r_{\mathcal{W}}, T_{S,M}), pk(\mathcal{S})),$
   $\mathcal{S}_{WRITE})$.

Let $Tr_{SCEP-\mathcal{WS}}$ be complementary to $Tr_{SCEP-\mathcal{SW}}$, such that $Tr_{SCEP-\mathcal{WS}}$ (1) includes Tr_{TLS-S} rather than Tr_{TLS-C}, and (2) inverts the directions of terms 2, 3, and 4.

Definition IX.7. We define two sets of "regular" strands that carry out the SCEP protocol:

1) Let $\Sigma_{SCEP-\mathcal{SW}}$ be the set of all strands with a trace of the form $Tr_{SCEP-\mathcal{SW}}$.
2) Let $\Sigma_{SCEP-\mathcal{WS}}$ be the set of all strands with a trace of the form $Tr_{SCEP-\mathcal{WS}}$.

Definition IX.8 (SCEP Strand Space). The SCEP strand space comprises $\Sigma_{pen} \cup \Sigma_{SCEP-\mathcal{SW}} \cup \Sigma_{SCEP-\mathcal{WS}}$.

B. Security Goals

We state formal security goals (Definitions IX.13, IX.14) for SCEP. To begin, we first formalize confidentiality (Definition IX.10) and agreement (Definition IX.11). These high-level security goals assert that, within a given strand space $\mathcal{P}$ and under an explicit set of *origination* assumptions, there exist no execution models (i.e., *shapes*), for which the specified properties do not hold. Later, we compose these definitions of confidentiality and agreement to define specific security goals for SCEP and the SecureDNA query and exemption protocols.

To prove confidentiality properties of execution models within a strand space $\mathcal{P}$, we define *listener strands* (Definition IX.9). To test the confidentiality of a sensitive value x, we define a listener strand with trace $Tr_{\mathcal{L}}(x)$. An execution model of $\mathcal{P}$ that satisfies $Tr_{\mathcal{L}}(x)$ illustrates a manner in which x leaks to the adversary. Thus, to prove confidentiality, we show

that there exists no listener strand $Tr_{\mathcal{L}}(x)$ in any complete execution of $\mathcal{P}$.

Definition IX.9 (Listener Strands). A *listener strand* for t is any strand with trace $Tr_{\mathcal{L}}(t) = [\, -t, +t\,]$, where $t \in A$ is an arbitrary term. Let $\Sigma_{\mathcal{L}}$ be the set of all listener strands.

Definition IX.10 (Confidentiality). Let $Tr_{\mathcal{X}}$ be any trace in $\mathcal{P}$. Let t be any term for which to test confidentiality. Let **orig** be the conjunction of origination predicates. Let $\vec{V}$ be a list of parameters of trace $Tr_{\mathcal{X}}$, such that the term t is an element of $\vec{V}$. For any strand $r \in \mathcal{P}$, $r : Tr$ indicates that r has trace Tr. Then,

$Conf_{\mathcal{P}}(Tr_{\mathcal{X}}, t, \textbf{orig}) \stackrel{\text{def}}{\Leftrightarrow}$
$(\forall r, l \in \mathcal{P} \cup \Sigma_{\mathcal{L}}, r : Tr_{\mathcal{X}}(\vec{V}) \wedge l : Tr_{\mathcal{L}}(t) \wedge \textbf{orig} \implies \perp).$

Definition IX.11 (Agreement). Let $Tr_{\mathcal{X}}$ be any trace in $\mathcal{P}$. Let $Tr_{\mathcal{Y}}$ be any trace in $\mathcal{P}$ such that $Tr_{\mathcal{X}} \neq Tr_{\mathcal{Y}}$. Let $\vec{C}$ be any list of values for which to test agreement. Let $\vec{V}_{\mathcal{X}}$ be a list of parameters of the trace $Tr_{\mathcal{X}}$ and $\vec{V}_{\mathcal{Y}}$ be a list of the parameters of trace $Tr_{\mathcal{Y}}$. Let **orig** be any conjunction of origination predicates. Then,

$Agree_{\mathcal{P}}(Tr_{\mathcal{X}}, Tr_{\mathcal{Y}}, \vec{C}, \textbf{orig}) \stackrel{\text{def}}{\Leftrightarrow}$
$(\forall r \in \mathcal{P}, r : Tr_{\mathcal{X}}(\vec{V}_{\mathcal{X}} \cup \vec{C}) \wedge \textbf{orig} \implies$
$\exists s \in \mathcal{P}, s : Tr_{\mathcal{Y}}(\vec{V}_{\mathcal{Y}} \cup \vec{C})).$

Definition IX.12 (SCEP Terms). Let $\vec{T}$ be the list of terms:

1) $\mathcal{S}, \mathcal{W}, M, CA \in Name$
2) $r_{\mathcal{S}}, r_{\mathcal{W}}, r'_{\mathcal{S}}, r'_{\mathcal{W}}, \omega \in Nonce$
3) $e_{\mathcal{S}}, e_{\mathcal{W}} \in Expt$.

We formalize security goals of SCEP from the perspective of $\mathcal{S}$ (Definition IX.13) and from the perspective of $\mathcal{W}$ (Definition IX.14). Each of these definitions specifies origination assumptions of the cryptographic perspective of $\mathcal{S}$ or $\mathcal{W}$, and comprises a conjunction of subgoals: (1) confidentiality of ω, and (2) agreement on the identities $\mathcal{S}$ and $\mathcal{W}$, the corresponding nonces $r_{\mathcal{S}}, r_{\mathcal{W}}$, and ω. Should both goals hold for all unique execution models in the SCEP strand space, we prove that ω does not leak to the adversary and that $\mathcal{S}$ and $\mathcal{W}$ achieve *injective agreement* [38] on the SCEP session parameters. Because of a crucial weakness in the SCEP (Section IX-C), we find SCEP is not "$\mathcal{W}$-$\mathcal{S}$" secure.

Definition IX.13 (SCEP $\mathcal{S} - \mathcal{W}$ Secure). Let **orig** be the conjunction of the following origination assumptions on $\vec{T}$:

1) $Non(sk(\mathcal{S}))$, $Non(sk(\mathcal{W}))$, $Non(sk(M))$, $Non(sk(CA))$, where Non indicates non-origination, and
2) $Uniq(\omega), Uniq(r_{\mathcal{S}}), Uniq(r'_{\mathcal{S}}), Uniq(e_{\mathcal{S}}), Uniq(e_{\mathcal{W}})$, where $Uniq$ indicates unique origination.

Uniquely originating terms are unknown to protocol participants and the network until a legitimate strand emits them as part of a message, enabling us to model random nonces, fresh secret keys, and other values that must be unique for each execution of a protocol. *Non-Originating* values are values such as private keys, which the adversary does not know,

cannot guess, and will never appear on the network in a decryptable form.

An SCEP strand space $\mathcal{P}$ is $\mathcal{S} - \mathcal{W}$ secure *if and only if (iff)* the conjunction of the following security goals holds:

1) $Conf_{\mathcal{P}}(Tr_{SCEP-\mathcal{SW}}(\vec{T}), \omega, \textbf{orig})$, and
2) $Agree_{\mathcal{P}}($
$Tr_{SCEP-\mathcal{SW}}(\vec{T}), Tr_{SCEP-\mathcal{WS}}(\vec{T}),$
$[\mathcal{S}, \mathcal{W}, r_{\mathcal{S}}, r_{\mathcal{W}}, \omega], \textbf{orig}).$

Definition IX.14 (SCEP $\mathcal{W} - \mathcal{S}$ Secure). Let **orig** be the conjunction of the following origination assumptions on $\vec{T}$:

1) $Non(sk(\mathcal{S}))$, $Non(sk(\mathcal{W}))$, $Non(sk(M))$, $Non(sk(CA))$, where Non indicates non-origination, and
2) $Uniq(\omega)$, $Uniq(r_{\mathcal{W}})$, $Uniq(r'_{\mathcal{W}})$, $Uniq(e_{\mathcal{S}})$, $Uniq(e_{\mathcal{W}})$), where $Uniq$ indicates unique origination.

An SCEP strand space $\mathcal{P}$ is $\mathcal{W} - \mathcal{S}$ secure iff the conjunction of the following security goals holds:

1) $Conf_{\mathcal{P}}(Tr_{SCEP-\mathcal{WS}}(\vec{T}), \omega, \textbf{orig})$, and
2) $Agree_{\mathcal{P}}(Tr_{SCEP-\mathcal{WS}}(\vec{T}),$
$Tr_{SCEP-\mathcal{SW}}(\vec{T}), [\mathcal{S}, \mathcal{W}, r_{\mathcal{S}}, r_{\mathcal{W}}, \omega], \textbf{orig}).$

C. Analysis

Using CPSA, we describe *all* minimal, essentially different execution models (shapes) within a strand space under a specific security goal's origination assumptions. Theorems IX.1 and IX.2 prove assertions about these SCEP security goals. A security goal holds iff it is true for all shapes possible within a strand space, which CPSA might verify by exhaustive search. A single counterexample is sufficient to disprove a security goal, when CPSA constructs a potential attack.

Theorem IX.1 (SCEP $\mathcal{S} - \mathcal{W}$ Security). If $\mathcal{P}$ is an SCEP strand space, then $\mathcal{P}$ is $\mathcal{S} - \mathcal{W}$ secure.

Proof by Enumeration. Carrying out an exhaustive search, CPSA enumerates all essentially different (unique) shapes under the $\mathcal{S} - \mathcal{W}$ security goal assumptions. None of these shapes contradict the security goal. □

Theorem IX.1 holds because $\mathcal{S}$ establishes a confidental and authenticated TLS channel with $\mathcal{W}$. As a result, the synthesizer believes ω to be confidential and agrees on ω and the nonces $r_{\mathcal{S}}, r_{\mathcal{W}}$ with the responder.

Theorem IX.2 (SCEP $\mathcal{W} - \mathcal{S}$ Security). If $\mathcal{P}$ is an SCEP strand space, then $\mathcal{P}$ is **not** $\mathcal{W} - \mathcal{S}$ secure.

Proof by Counterexample. Using CPSA, we identify all unique shapes that contradict the confidentiality and agreement subgoals.

Confidentiality. In each shape, a listener strand learns the value of ω when $\mathcal{W}$ communicates with penetrator strands masquerading as a legitimate client.

Agreement. In each shape, a legitimate strand with the trace of $\mathcal{W}$ fails to agree on the values of $\omega, r_{\mathcal{S}}, r_{\mathcal{W}}$ with any regular strand. □

In counterexamples that prove Theorem IX.2, penetrator strands establish parallel TLS connections with the legitimate $\mathcal{W}$ and $\mathcal{S}$. Consequently, an adversary is able to break the confidentiality and agreement subgoals. This failure results from (1) the one-way authentication of TLS, which only authenticates the server to the client, and (2) SCEP's failure to mutually authenticate $\mathcal{S}$ and $\mathcal{W}$.

SCEP fails to mutually authenticate because it fails to make explicit the intended recipients of Messages (3) and (4) in the trace (Definition IX.6), enabling a MitM attack. This flaw is similar to that of the 1978 NS public-key protocol. Due to this weakness, composing SCEP with TLS produces an outcome no more secure than one-way authenticated TLS, which fails to satisfy the SCEP goal of mutual authentication. In Section X, we propose and formally verify improvements to the SCEP that enable mutual authentication, and thus $\mathcal{W} - \mathcal{S}$ security.

X. Suggested Improvements, Including to SCEP

We propose and verify a correction to the SCEP that satisfies our formal security goals (Section IX-B). In summary, the SCEP fails to mutually authenticate because it does not bind critical values (token of the other party, request cookie) to the hash that each communicant signs and transmits. The result is that $\mathcal{S}$ is certain of the identity of $\mathcal{W}$ (because SCEP runs within a one-way authenticated TLS session), but $\mathcal{W}$ has no guarantee that the cookie ω remains confidential, and $\mathcal{W}$ cannot determine that it communicates with any legitimate instance of $\mathcal{S}$. To correct this error, we modify the SCEP traces in Section IX-A and repeat our analysis on the resulting strand space for our improvement, which we call SCEP+.

First, we update the SCEP traces (Definition IX.6) to include additional information in the hashes by incorporating both communicant tokens and the cookie into each hash (Definition X.1). Each party signs its hash using a key unknown to the adversary. This step binds ω and the tokens $T_{\mathcal{S},M}^M, T_{\mathcal{W},\mathcal{CA}}^{IF}$ within a single SCEP session.

Definition X.1 (SCEP+ Traces)**.** Let $Tr_{SCEP+-\mathcal{SW}}$ and $Tr_{SCEP+-\mathcal{WS}}$ modify the traces $Tr_{SCEP-\mathcal{SW}}$ and $Tr_{SCEP-\mathcal{WS}}$ (Definition IX.6) with the alterations:

1) $h(\text{``server-mutauth''}, r_{\mathcal{S}}, r_{\mathcal{W}}, \omega, T_{\mathcal{S},M}, T_{\mathcal{W},\mathcal{CA}})$
2) $h(\text{``client-mutauth''}, r_{\mathcal{S}}, r_{\mathcal{W}}, \omega, T_{\mathcal{S},M}, T_{\mathcal{W},\mathcal{CA}})$.

Next, we update the definitions of SCEP $\mathcal{S} - \mathcal{W}$ (Definition IX.13) and $\mathcal{W} - \mathcal{S}$ (Definition IX.14) security with the $SCEP+$ traces. Because we are not adding any terms, the security goals do not change. We refer to these security goals as $SCEP+$ $\mathcal{S} - \mathcal{W}$ and $SCEP+$ $\mathcal{W} - \mathcal{S}$.

Finally, we state and prove two new theorems that assert confidentiality of ω and agreement on the session parameters for SCEP+. Because SCEP+ includes sufficient encrypted information in each message, both parties are able to authenticate each other. As a result, we prove Theorem X.1 and Theorem X.2 using CPSA.

Theorem X.1 (SCEP+ $\mathcal{S} - \mathcal{W}$ Security)**.** If $\mathcal{P}$ is an SCEP+ strand space, then $\mathcal{P}$ is $\mathcal{S} - \mathcal{W}$ secure.

Proof by Enumeration. There exists no shape that contradicts $\mathcal{S} - \mathcal{W}$ security. □

Theorem X.2 (SCEP+ $\mathcal{W} - \mathcal{S}$ Security)**.** If $\mathcal{P}$ is an SCEP+ strand space, then $\mathcal{P}$ is $\mathcal{W} - \mathcal{S}$ secure.

Proof by Enumeration. There exists no shape that contradicts $\mathcal{W} - \mathcal{S}$ security. □

Our improvement, SCEP+, does not require any additional cryptographic calls and requires minimal changes. SCEP+ achieves SCEP's goal of mutual authentication. Software release 1.1.0 implements our recommended SCEP+ protocol, which change involved approximately five lines of code.

XI. Discussion

Our study highlights that a secure system requires more than sound abstract mathematical cryptography and UC proofs; a secure system also needs careful attention to design, engineering, implementation, usability, key management, and procedures, and careful attention to many associated details. Useful, clever, novel cryptography underlies the SecureDNA system [5], but to achieve its security goals, the system also depends on its protocols, system design, and implementation.

The way the SecureDNA system deals with authenticated channels is problematic and inconsistent. Communications are initiated with one-way TLS, not mTLS. In addition to TLS, SecureDNA uses its own flawed custom mutual authentication protocol SCEP. If SecureDNA trusts TLS, then deploying an additional authentication protocol would add unnecessary complexity. If SecureDNA does not trust TLS, then it should use a strong protocol that performs effective cryptographic bindings (see Section XII-B).

Our analysis of SecureDNA highlights several important security engineering principles that can serve as lessons learned for others. (1) Cryptographically bind protocol messages to their full context so that they cannot be misused out of context. Almost all known structural weaknesses of protocols violate this essential principle. (2) Favor proven standard solutions over custom protocols. Carefully document and justify deviations from this principle. (3) When designing protocols, work throughout the process with experts in formal-methods analysis to ensure that appropriate security goals are achieved. (4) Be aware that secure channels might fail at the PKI level for a variety of reasons, and where appropriate, add a second layer of defense at the application layer.

Although it would be impossible for us to know all of the rationale behind the SecureDNA team's design choices, we understand that their decision not to use mTLS was interwoven with their decision not to use X.509. They explained that they avoided using X.509 because of its complexity, failure to support threshold roots, and limited support in the Rust ecosystem. In making this engineering tradeoff, they assumed greater risk in the subtle non-trivial task of designing their own mutual authentication protocol.

SecureDNA relies heavily on *security operations center (SOC)* software and procedures against rate-limiting attacks

and other malicious behaviors. Although there is a role for such monitoring, it would be better where possible to strengthen the protocols rather than to rely on SOC as a first line of defense. Even if auditing procedures are currently meticulously followed, there is no assurance that they will continue to be properly followed in the future. Experience shows that effective monitoring is expensive and difficult to maintain over long periods of time. There is no assurance that the SOC software is free of errors, and all controls must be sensitive to differences among providers. Although SecureDNA's SOC provides significant protection against our rate-limiting attacks, better protocol design and engineering practices might reduce the risk of human error in these areas.

The SecureDNA protocols depend strongly on the confidentiality, authentication, and integrity properties of the TLS channel. In some settings, however, TLS channels can be defeated due to poor management of root TLS certificates or vulnerabilities with a corporate firewall that acts as a MitM TLS proxy to monitor traffic [41]. SecureDNA cannot control the security practices of the providers.

The structural vulnerabilities we uncovered in the protocols might have been avoided if formal-methods analysis had been used throughout the design and evaluation process. It would have been helpful to state security goals formally, to construct formal protocol models, and to perform formal-methods analyses on those models. It would have been helpful to collaborate with experts who have the capability to perform such formal-methods analyses.

Open problems include a security review of the software implementation, including the auditing capabilities. For example, the SecureDNA system deploys a web form through which $\mathcal{B}$ applies for an ELT. Vulnerabilities are very common in such forms, and a vulnerability in this form might enable attacks that circumvent $\mathcal{B}$. Given that SecureDNA depends critically on TLS, it would be prudent to perform a thorough security review of its implementation and integration of TLS.

XII. FINDINGS AND RECOMMENDATIONS

We summarize our major findings of vulnerabilities and offer recommendations for mitigating them.

A. Findings

(1) The custom SCEP protocol achieves only one-way authentication. This structural weakness enables the adversary to circumvent rate limits and mount DoS attacks, if $\mathcal{S}$ connects with a malicious or corrupted $\mathcal{H}$ or $\mathcal{K}$.

(2) The system lacks adequate cryptographic bindings of certificates, tokens, and responses, including of responses from $\mathcal{H}$ to $\mathcal{S}$. This undesirable structural weakness prevents the system from detecting if responses, within a TLS channel, from the hazards database were modified. If a synthesizer were to reconnect with $\mathcal{H}$ over the same TLS session (which the implementation disallows), the adversary could replay and swap responses from $\mathcal{H}$ without breaking TLS.

(3) Appendix B points out several additional security issues, including providers possibly storing passphrases in plain text

files, and depending in part on the assurance of email for auditing alerts and records. These examples illustrate difficult challenges stemming from engineering tradeoffs, limited options, and lack of control over provider actions.

(4) From discussions with the SecureDNA team, we learned that part of SecureDNA's security strategy depends significantly, not on a model of prevention, but on a model of monitor, detect, and respond. SecureDNA carries out this strategy using non-publicly released source code (unavailable to us) and a combination of automatic and manual interventions.

B. Recommendations

To mitigate the security vulnerabilities we identified, we recommend the following:

(1) To mitigate the vulnerabilities from Section VIII, instead of using the custom authentication protocol SCEP, use mTLS [37]. Given that SecureDNA provides certificate hierarchies, it would be relatively simple to do so. Similarly, instead of using a custom certificate structure, start with a standard one such as X.509 [42] (see also Appendix B). In the alternative, as newly released Version 1.1.0 does, replace SCEP with our suggested SCEP+, as explained in Section X.

(2) In each of the protocols, strengthen the cryptographic binding of messages to context. For example, bind each response from $\mathcal{H}$ to the associated query from $\mathcal{S}$. Also, bind messages and tokens to the channels in which they are used (e.g., see [25], [43]).

(3) Perform a thorough security review of SecureDNA's implementation and integration of TLS.

(4) As is true for most systems, it is important to devote great attention to detailed issues of system design and operations, including those identified in Appendix B. For example, SecureDNA does recommend to providers that they store keys appropriately, including in TPMs [44] when suitable, but SecureDNA cannot control such practices.

XIII. CONCLUSION

Our main findings are the two structural weaknesses (one-way authentication and inadequate cryptographic bindings) summarized in Section XII-A. Even if these attacks pose low risks, including due to monitoring, eliminating the underlying structural weaknesses would strengthen the system. Other attacks might be possible.

SG2 (secrecy of the query) appears solidly protected by the blinding mechanisms, and this goal does not depend on the strength of the TLS implementation and integration. The other security goals depend critically on TLS, and the security engineering devoted to protecting them is less convincing.

We prove that our improved SCEP+ satisfies our precisely-stated security goals (including mutual authentication) in the DY model. Release 1.1.0 implements our suggestion.

Our study demonstrates that secure systems require more than sound abstract mathematical cryptography. Our study also highlights tensions between security and fielding a usable system that providers will adopt. Although a malicious entity could avoid controls by using their own synthesis

machine, or using a synthesis provider that does not screen, the SecureDNA system can meaningfully raise the safety of legitimate synthesis labs. We hope that our analysis and suggested mitigations will strengthen the SecureDNA system.

ACKNOWLEDGMENTS

REFERENCES

[1] R. Barrangou and J. A. Doudna, "Applications of CRISPR technologies in research and beyond," *Nature biotechnology*, vol. 34, no. 9, pp. 933–941, 2016.

[2] D. Kokotajlo, S. Alexander, T. Larsen, and E. L. R. Dean, "AI 2027." https://ai-2027.com/, April 2025.

[3] C. Baum, J. Berlips, W. Chen, H. Cui, I. Damgård, J. Dong, K. M. Esvelt, L. Foner, M. Gao, D. Gretton, M. Kysel, J. Li, X. Li, O. Paneth, R. L. Rivest, F. Sage-Ling, A. Shamir, Y. Shen, M. Sun, V. Vaikuntanathan, L. Van Hauwe, T. Vogel, B. Weinstein-Raun, Y. Wang, D. Wichs, S. Wooster, A. C. Yao, Y. Yu, H. Zhang, and K. Zhang, "A system capable of verifiably and privately screening global DNA synthesis," *arXiv preprint arXiv:2403.14023*, 2024.

[4] SecureDNA Foundation, "SecureDNA website." https://securedna.org/, 2025.

[5] C. Baum, J. Berlips, W. Chen, I. B. Damgård, K. M. Esvelt, L. Foner, D. Gretton, M. Kysel, R. L. Rivest, L. Roy, F. Sage-Ling, A. Shamir, V. Vaikuntanathan, L. V. Hauwe, T. Vogel, B. Weinstein-Raun, D. Wichs, S. Wooster, A. C. Yao, and Y. Yu, "Efficient maliciously secure oblivious exponentiations," *IACR Communications in Cryptology*, vol. 1, no. 3, 2024.

[6] C. Baum, H. Cui, I. Damgård, K. Esvelt, M. Gao, D. Gretton, O. Paneth, R. Rivest, V. Vaikuntanathan, D. Wichs, *et al.*, "Cryptographic aspects of DNA screening." https://people.csail.mit.edu/rivest/pubs/BE20.pdf, 2020.

[7] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[8] D. R. Stinson, "An explication of secret sharing schemes," *Designs, Codes and Cryptography*, vol. 2, no. 4, pp. 357–390, 1992. 10.1007/BF00125203.

[9] SecureDNA Foundation, "SecureDNA source code." https://github.com/SecureDNA, 2025.

[10] M. D. Liskov, J. D. Ramsdell, J. D. Guttman, and P. D. Rowe, *The Cryptographic Protocol Shapes Analyzer: A Manual*, 2016.

[11] R. Canetti, "Universally composable security," *Journal of the ACM (JACM)*, vol. 67, no. 5, pp. 1–94, 2020.

[12] G. Lowe, "An attack on the Needham-Schroeder Public-Key Authentication Protocol," *Inf. Process. Lett.*, vol. 56, no. 3, pp. 131–133, 1995.

[13] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," *Commun. ACM*, vol. 21, pp. 993–999, Dec. 1978.

[14] E. Rescorla, *SSL and TLS: Designing and Building Secure Systems*. Addison-Wesley, 2001.

[15] E. Rescorla and T. Dierks, "The Transport Layer Security (TLS) Protocol Version 1.2." RFC 5246, Aug. 2008.

[16] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3." RFC 8446, Aug. 2018.

[17] Anonymous, "SecureDNA CPSA Model Repository." https://anonymous.4open.science/r/securedna-cpsa-anon-BB46/README.md, July 2025.

[18] A. T. Sherman, J. J. R. Romano, E. Zieglar, E. Golaszewski, and J. D. Fuchs, "Analysis of the security design, engineering, and implementation of the SecureDNA system," *arXiv preprint arXiv:2512.09233*, 2025.

[19] M. Langenkamp, A. Lin, A. Quach, and G. Hu, "Improving the securedna system." Course project for 6.857 at MIT, 2022.

[20] A. Kane and M. T. Parker, "Screening state of play: the biosecurity practices of synthetic dna providers," *Applied Biosafety*, vol. 29, no. 2, pp. 85–95, 2024.

[21] S. A. Hoffmann, J. Diggans, D. Densmore, J. Dai, T. Knight, E. Leproust, J. D. Boeke, N. Wheeler, and Y. Cai, "Safety by design: Biosafety and biosecurity in the age of synthetic genomics," *Iscience*, vol. 26, no. 3, 2023.

[22] N. S. A. B. for Biosecurity, "Addressing biosecurity concerns related to the synthesis of select agents." National Institutes of Health, Bethesda, MD, 2006.

[23] I. G. S. Consortium, "Harmonized screening protocol v2.0." https://genesynthesisconsortium.org/wp-content/uploads/IGSCHarmonizedProtocol11-21-17.pdf, 2017.

[24] J. H. B. S. of Public Health Center for Health Science, "Gene synthesis screening information hub." https://genesynthesisscreening.centerforhealthsecurity.org/.

[25] E. Golaszewski, E. Zieglar, A. T. Sherman, K. Abou Elsaad, and J. D. Fuchs, "Limitations of wrapping protocols and TLS channel bindings: Formal-methods analysis of the Session Binding Proxy protocol," in *International Conference on Research in Security Standardisation (SSR 2024)*, pp. 81–119, Springer, 2024.

[26] B. Blanchet, "Automatic verification of security protocols in the symbolic model: The verifier ProVerif," in *Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures* (A. Aldini, J. López, and F. Martinelli, eds.), vol. 8604 of *Lecture Notes in Computer Science*, pp. 54–87, Springer, 2013.

[27] S. Meier, B. Schmidt, C. Cremers, and D. A. Basin, "The TAMARIN prover for the symbolic analysis of security protocols," in *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings* (N. Sharygina and H. Veith, eds.), vol. 8044 of *Lecture Notes in Computer Science*, pp. 696–701, Springer, 2013.

[28] S. Escobar, C. Meadows, and J. Meseguer, "Maude-NPA: Cryptographic protocol analysis modulo equational properties," in *Foundations of Security Analysis and Design V, FOSAD 2007/2008/2009 Tutorial Lectures* (A. Aldini, G. Barthe, and R. Gorrieri, eds.), vol. 5705 of *Lecture Notes in Computer Science*, pp. 1–50, Springer, 2007.

[29] M. D. Liskov, J. D. Ramsdell, J. D. Guttman, and P. D. Rowe, *The Cryptographic Protocol Shapes Analyzer: A Manual*. The MITRE Corporation, 2016.

[30] F. J. Thayer, J. C. Herzog, and J. D. Guttman, "Strand spaces: Proving security protocols correct," *J. Comput. Secur.*, vol. 7, no. 1, pp. 191–230, 1999.

[31] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.

[32] M. Liskov, P. Rowe, and J. Thayer, "Completeness of CPSA," tech. rep., MITRE, 2011. https://www.mitre.org/sites/default/files/pdf/12_0038.pdf.

[33] P. D. Rowe, J. D. Guttman, and M. D. Liskov, "Measuring protocol strength with security goals," *Int. J. Inf. Sec.*, vol. 15, no. 6, pp. 575–596, 2016.

[34] A. Malpani, S. Galperin, and C. Adams, "X.509 Internet public key infrastructure online certificate status protocol-ocsp," tech. rep., IETF, 2013. RFC 6960, 2013.

[35] C. Cartas, "Rust – The programming language for every industry," *Economy Informatics Journal*, vol. 19, pp. 45–51, 09 2019.

[36] D. Boneh, "The decision Diffie-Hellman problem," in *International algorithmic number theory symposium*, pp. 48–63, Springer, 1998.

[37] B. Campbell, J. Bradley, N. Sakimura, and T. Lodderstedt, "RFC 8705: Oauth 2.0 mutual-tls client authentication and certificate-bound access tokens," 2020.

[38] G. Lowe, "A hierarchy of authentication specifications," in *10th Computer Security Foundations Workshop Proceedings*, pp. 31–43, IEEE CS Press, 1997.

[39] B. Harris and R. Hunt, "TCP/IP security threats and attack methods," *Computer Communications*, vol. 22, no. 10, pp. 885–897, 1999.

[40] W. Burgers, R. Verdult, and M. C. J. D. van Eekelen, "Prevent session hijacking by binding the session to the cryptographic network credentials," in *Secure IT Systems - 18th Nordic Conference, NordSec 2013, Ilulissat, Greenland, October 18-21, 2013, Proceedings* (H. R. Nielson

and D. Gollmann, eds.), vol. 8208 of *Lecture Notes in Computer Science*, pp. 33–50, Springer, 2013.

[41] M. O'Neill, S. Ruoti, K. Seamons, and D. Zappala, "Tls proxies: Friend or foe?," in *Proceedings of the 2016 Internet Measurement Conference*, pp. 551–557, 2016.

[42] "X.509 : Information technology - open systems interconnection - the directory: Public-key and attribute certificate frameworks recommendation X.509." https://www.itu.int/rec/T-REC-X.509.

[43] E. Golaszewski, A. T. Sherman, and E. Zieglar, "Cryptographic binding should not be optional: A formal-methods analysis of FIDO UAF authentication," in *International Conference on Research in Security Standardisation (SSR 2025)*, Lecture Notes in Computer Science, Springer, December 2025. in press.

[44] Trusted Computing Group (TCG), "TPM 2.0 Library." https://trustedcomputinggroup.org/resource/tpm-library-specification/, 2025.

[45] J. R. Romano, S. Bokka, Z. Heck, T. Caplan, W. Zheng, and S. Stultz, "Modeling and formal analysis of authentication in the SecureDNA protocol." CMSC-491/691: Cybersecurity Research (INSuRE), University of Maryland, Baltimore County, December 2024.

[46] B. Guest, I. Roland-Reid, T. R. Walke, M. Asatkar, and T. Sarkar, "Security analysis of registration, exemption handling, and web app security of the SecureDNA protocol." CMSC-491/691: Cybersecurity Research (INSuRE), University of Maryland, Baltimore County, December 2024.

[47] A. T. Sherman, M. Dark, A. Chan, T. Morris, L. Oliva, J. Springer, B. Thuraisingham, C. Vatcher, R. Verma, and S. Wetzel, "The INSuRE Project: CAE-Rs collaborate to engage students in cybersecurity research," *IEEE Security & Privacy*, vol. 15, Aug. 2017.

[48] "Information Security Research and Education (INSuRE)." https://caecommunity.org/initiative/insure. [Online; accessed 15-May-2025].

[49] J. R. Romano, "Analysis of SecureDNA system key management." CMSC-652 Cryptology, University of Maryland, Baltimore County, May 2025.

[50] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive secret sharing or: How to cope with perpetual leakage," in *Advances in Cryptology—CRYPT0'95: 15th Annual International Cryptology Conference Santa Barbara, California, USA, August 27–31, 1995 Proceedings 15*, pp. 339–352, Springer, 1995.

[51] D. J. Bernstein and T. Lange, "Post-quantum cryptography," *Nature*, vol. 549, no. 7671, pp. 188–194, 2017.

APPENDIX

A. Ethical Considerations

We have responsibly disclosed our findings to the SecureDNA team by sharing preliminary drafts of our paper, exchanging emails, and meeting with them remotely. These meetings took place on June 3, 2025, and July 30, 2025. Newly released Version 1.1.0 of the SecureDNA system fixes SCEP with our proposed SCEP+ protocol. It also implements an optional "verifiable screening" mode, which can mitigate the response-switching attack.

B. Other Security Issues

This section briefly identifies several additional security design issues of the SecureDNA system we reviewed that are noteworthy of consideration, even if they do not present imminent high risk. Some of these issues highlight tradeoffs inherent in engineering systems, including tradeoffs between security strength and user requirements.

(1) The published descriptions [3] and source code of the SecureDNA system do not describe how they would replace the shared DOPRF key k should it ever be compromised. All security systems should address re-keying procedures. The SecureDNA team explained that, should they need to replace k, they would do what they do whenever they regenerate $\mathcal{H}$: generate a new k and regenerate $\mathcal{H}$, which they say they can do quickly.

(2) In the current system implementation, the DOPRF key is generated *centrally*, not in a *distributed* fashion as per their design, creating a single point of failure. The SecureDNA team stated that a future update will address this issue.

(3) Some intended potentially important security features were not yet implemented at the time of our review. These features include *proactive* secret sharing [50] (in which key shares are rotated to mitigate the threat that an adversary might eventually compromise all keyservers, a few at a time). The SecureDNA team explained that, instead of rotating the key shares, they can re-key and regenerate $\mathcal{H}$. They also explained that a new version of the code now checks for high rate limits, and that they monitor and enforce rate limits at servers when clients make requests.

(4) The system encrypts stored keys with user-entered passphrases, but writes these passphrases as plain text to a provider-specified location. This location might be an ordinary file, possibly defeating any benefit from encrypting the keys. While DNA synthesis providers have the option to institute stronger security practices (e.g., involving a password manager or TPM), these practices are not mandated and SecureDNA cannot control them.

(5) Some of the auditing and record-keeping mechanisms depend in part on the integrity and assured operation of email. For example, the exemption-handling protocol sends an alert to $\mathcal{B}$ via email whenever it processes an ELT [E24]. A powerful attacker might be able to block email notifications. The SecureDNA team explained that providers and $\mathcal{B}$ typically use cloud-based email services and that all of the auditing emails are encrypted.

(6) Rather than using a well-vetted library to support certificate infrastructure, SecureDNA creates its own custom infrastructure, unnecessarily increasing complexity and risk of security errors. Given that the X.509 standard [42] includes an optional user-defined field, we feel there is no compelling reason to create a custom infrastructure (see also Section XI).

(7) In the basic-request protocol, $\mathcal{S}$'s response to $\mathcal{C}$ includes information that is potentially dangerous, especially for novel hazards: if the request is denied, $\mathcal{S}$ includes in its response which sequence(s) in the requested list of sequences are hazardous. It also states the pathogen from which the hazardous sequence came and why that pathogen is dangerous. The SecureDNA team explained that clients and providers demand this information, and novel hazards are treated differently.

(8) The implemented DOPRF is not post-quantum secure [51]. The SecureDNA team doubted that any post-quantum secure DOPRF would meet their time and space performance requirements. They also said that a post-quantum adversary would be able to break TLS.

C. Summary Results from Formal-Methods Analyses of SecureDNA Protocols

TABLE I

SECURITY PROPERTIES OF SCEP AND SCEP+. FOR EACH MODEL, A CHECK (✓) INDICATES THAT, FOR ALL POSSIBLE EXECUTION MODELS UNDER CORRESPONDING GOAL'S SECURITY ASSUMPTIONS, THE LOGICAL SECURITY GOAL HOLDS. A CROSSMARK (×) INDICATES THAT CPSA FINDS A COUNTEREXAMPLE THAT DISPROVES THE CORRESPONDING SECURITY GOAL. SEE DEFINITION IX.13 AND DEFINITION IX.14 FOR THE SECURITY GOAL DEFINITIONS.

Model	$\mathcal{S}$-$\mathcal{W}$ Secure	$\mathcal{W}$-$\mathcal{S}$ Secure
SCEP	✓	×
SCEP+	✓	✓

TABLE II

SECURITY PROPERTIES OF BASIC QUERY AND EXEMPTION QUERY. TO ACHIEVE CONFIDENTIALITY OF $M(s)$ AND AGREEMENT ON $M(s)^k$, $M(s^E)$, $AuthCode$, $Resp$, THE PROTOCOL MUST SATISFY THE CORRESPONDING SECURITY GOAL DEFINITIONS IN APPENDIX B (BASIC QUERY PROTOCOL) OF [18] AND APPENDIX C (EXEMPTION QUERY PROTOCOL) OF [18].

Model	Confidentiality of $M(s)$	Agreement on $M(s)^k$, $M(s^E)$, $AuthCode$, $Resp$
Basic Query (No TLS)	✓	×
Basic Query	✓	×
Exemption Query	✓	×
Basic Query (SCEP+)	✓	✓

D. Rate-Limiting Attack by Corrupt Keyserver

We provide more details about the rate-limiting attack described in Section VII-C. Whenever a legitimate synthesizer $\mathcal{S}$ makes a request to a corrupt keyserver $\mathcal{K}'$, the following rate-limiting attack presents itself to $\mathcal{K}'$. A similar attack is possible if $\mathcal{S}$ connects with a corrupt $\mathcal{H}$.

[1] $\mathcal{S} \rightarrow \mathcal{K}'$: Nonce($\mathcal{S}$), Token($\mathcal{S}$)

Following this exchange, $\mathcal{K}'$ now knows Nonce($\mathcal{S}$) and possesses Token($\mathcal{S}$). Assume that $\mathcal{S}$ is far below their current rate limit. $\mathcal{K}$ now has the opportunity to masquerade as $\mathcal{S}$ to other keyservers or to $\mathcal{H}$. Let $\mathcal{W}$ denote any legitimate responding infrastructure (e.g., $\mathcal{H}$).

[2] $\mathcal{K}' \rightarrow \mathcal{W}$: Nonce($\mathcal{S}$), Token($\mathcal{S}$)
[3] $\mathcal{K}' \leftarrow \mathcal{W}$: Cookie, Nonce($\mathcal{W}$), Token($\mathcal{W}$), x = Sign[Hash("server-mutauth", Nonce($\mathcal{S}$), Nonce($\mathcal{W}$), Token($\mathcal{W}$)), Privk($\mathcal{W}$)]

$\mathcal{K}'$ also knows Nonce($\mathcal{W}$) and Token($\mathcal{W}$), and has a signed copy of the hash that $\mathcal{S}$ expects. $\mathcal{K}'$ can now exploit the weakness of SCEP to trick $\mathcal{S}$ into authenticating $\mathcal{K}'$ to $\mathcal{W}$.

[4] $\mathcal{S} \leftarrow \mathcal{K}'$: Cookie, Nonce($\mathcal{W}$), Token($\mathcal{W}$), x
[5] $\mathcal{S} \rightarrow \mathcal{K}'$: Cookie, y = Sign[Hash("client-mutauth", Nonce($\mathcal{S}$), Nonce($\mathcal{W}$), Token(S)), Privk($\mathcal{S}$)]

Step 5 is important: $\mathcal{K}'$ obtains the critical value y. Knowledge of y enables $\mathcal{K}'$ to masquerade as $\mathcal{S}$ to $\mathcal{W}$.

[6] $\mathcal{K}' \rightarrow \mathcal{W}$: Cookie, y

$\mathcal{W}$ now believes $\mathcal{K}'$ is $\mathcal{S}$. Thus, $\mathcal{K}'$ is able to use the rate limit budget of $\mathcal{S}$ for their own requests to other keyservers or $\mathcal{H}$. Behold $\mathcal{K}'$ can perform this attack on many different instances of $\mathcal{S}$ simultaneously. This attack is within SecureDNA's adversarial model, but Baum et al. [5], [6] did not consider rate-limiting attacks.
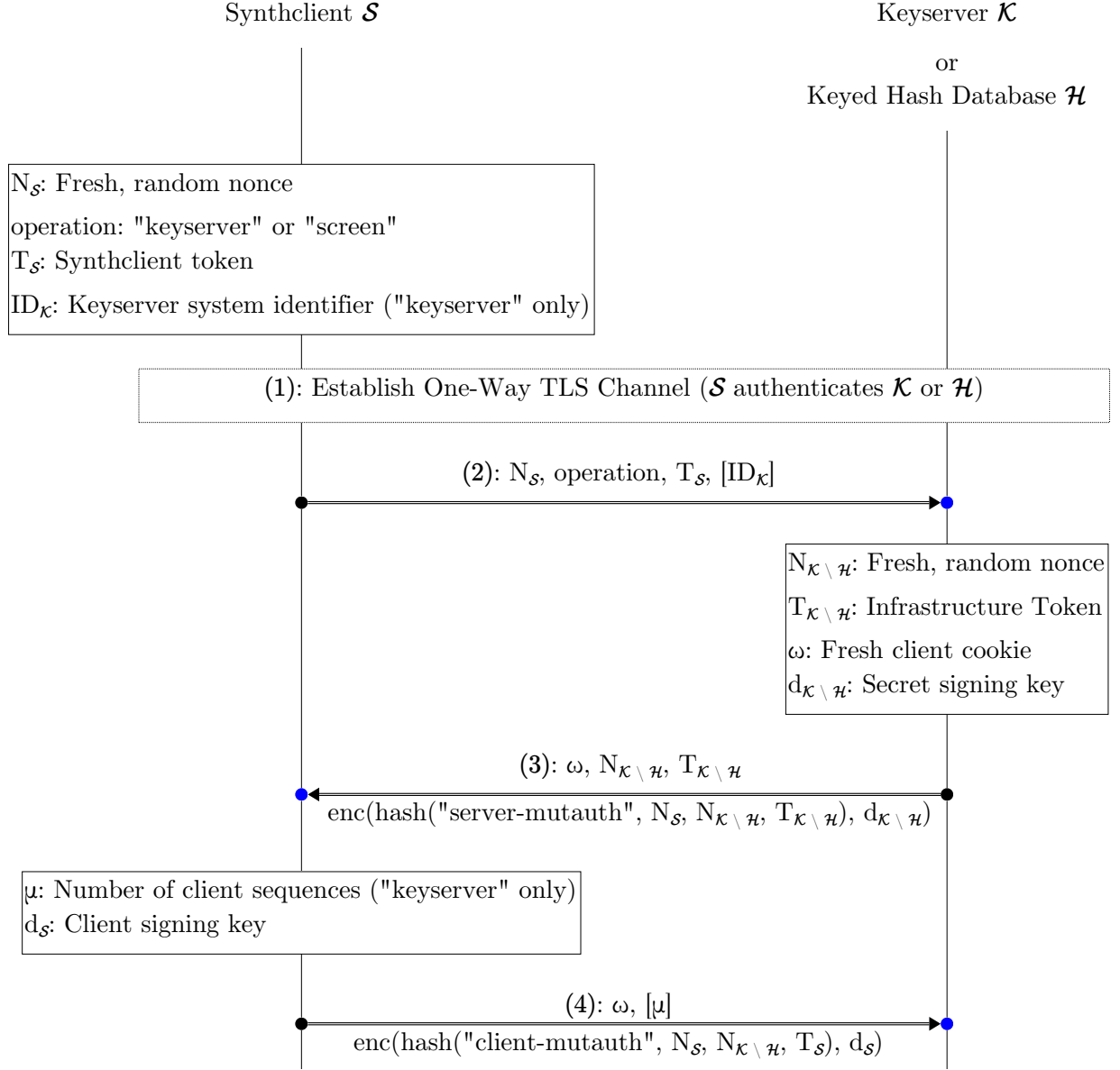
E. Ladder Diagrams of SecureDNA Protocols

Synthclient $\mathcal{S}$ Keyserver $\mathcal{K}$
                                                                        or
                                                            Keyed Hash Database $\mathcal{H}$

N$_\mathcal{S}$: Fresh, random nonce

operation: "keyserver" or "screen"
T$_\mathcal{S}$: Synthclient token

ID$_\mathcal{K}$: Keyserver system identifier ("keyserver" only)

(1): Establish One-Way TLS Channel ($\mathcal{S}$ authenticates $\mathcal{K}$ or $\mathcal{H}$)

(2): N$_\mathcal{S}$, operation, T$_\mathcal{S}$, [ID$_\mathcal{K}$]

N$_{\mathcal{K} \setminus \mathcal{H}}$: Fresh, random nonce

T$_{\mathcal{K} \setminus \mathcal{H}}$: Infrastructure Token

ω: Fresh client cookie
d$_{\mathcal{K} \setminus \mathcal{H}}$: Secret signing key

(3): ω, N$_{\mathcal{K} \setminus \mathcal{H}}$, T$_{\mathcal{K} \setminus \mathcal{H}}$

enc(hash("server-mutauth", N$_\mathcal{S}$, N$_{\mathcal{K} \setminus \mathcal{H}}$, T$_{\mathcal{K} \setminus \mathcal{H}}$), d$_{\mathcal{K} \setminus \mathcal{H}}$)

μ: Number of client sequences ("keyserver" only)
d$_\mathcal{S}$: Client signing key

(4): ω, [μ]
enc(hash("client-mutauth", N$_\mathcal{S}$, N$_{\mathcal{K} \setminus \mathcal{H}}$, T$_\mathcal{S}$), d$_\mathcal{S}$)

Fig. 4. Ladder diagram of the custom mutual authentication *(SCEP)* protocol. Vertical lines correspond to communicating roles ($\mathcal{S}$, $\mathcal{K}$, $\mathcal{H}$). Arrows between vertical lines indicate roles transmitting and receiving protocol messages. SCEP assumes an existing one-way authenticated TLS channel between the synthesizer $\mathcal{S}$, and the keyserver $\mathcal{K}$ or the keyed hash database $\mathcal{H}$. Some message components appear only when $\mathcal{S}$ communicates with $\mathcal{K}$; we indicate these components by surrounding them with brackets ([]). The responder nonce N, token T, and secret signing key d correspond to the entity with which $\mathcal{S}$ communicates. Upon completing SCEP, $\mathcal{S}$ obtains a request cookie ω, which $\mathcal{S}$ transmits in a subsequent request to $\mathcal{K}$ or $\mathcal{H}$.

Synthesizer $\mathcal{S}$ Keyserver $\mathcal{K}$ Keyed Hashed Database $\mathcal{H}$

(1): SCEP("keyserver", ID $_\mathcal{K}$)

$\omega_\mathcal{K}$: Keyserver cookie

s: Client sequences
s_E: Client exemption sequences
β: Random blinding value
$M(s)^x$: DOPRF under key x

(2): $\omega_\mathcal{K}$, $M(s)^\beta$, $[M(s_E)^\beta]$

k: Keyserver key

(3): $M(s)^{\beta k}$, $[M(s_E)^{\beta k}]$

(4): SCEP("screen")

$\omega_\mathcal{H}$: Database cookie

T^E: Exemption token
$O_\mathcal{A}$: Auth device code

(5) $\omega_\mathcal{H}$, $M(s)^k$, $[T^E, O_\mathcal{A}, M(s_E)^k]$
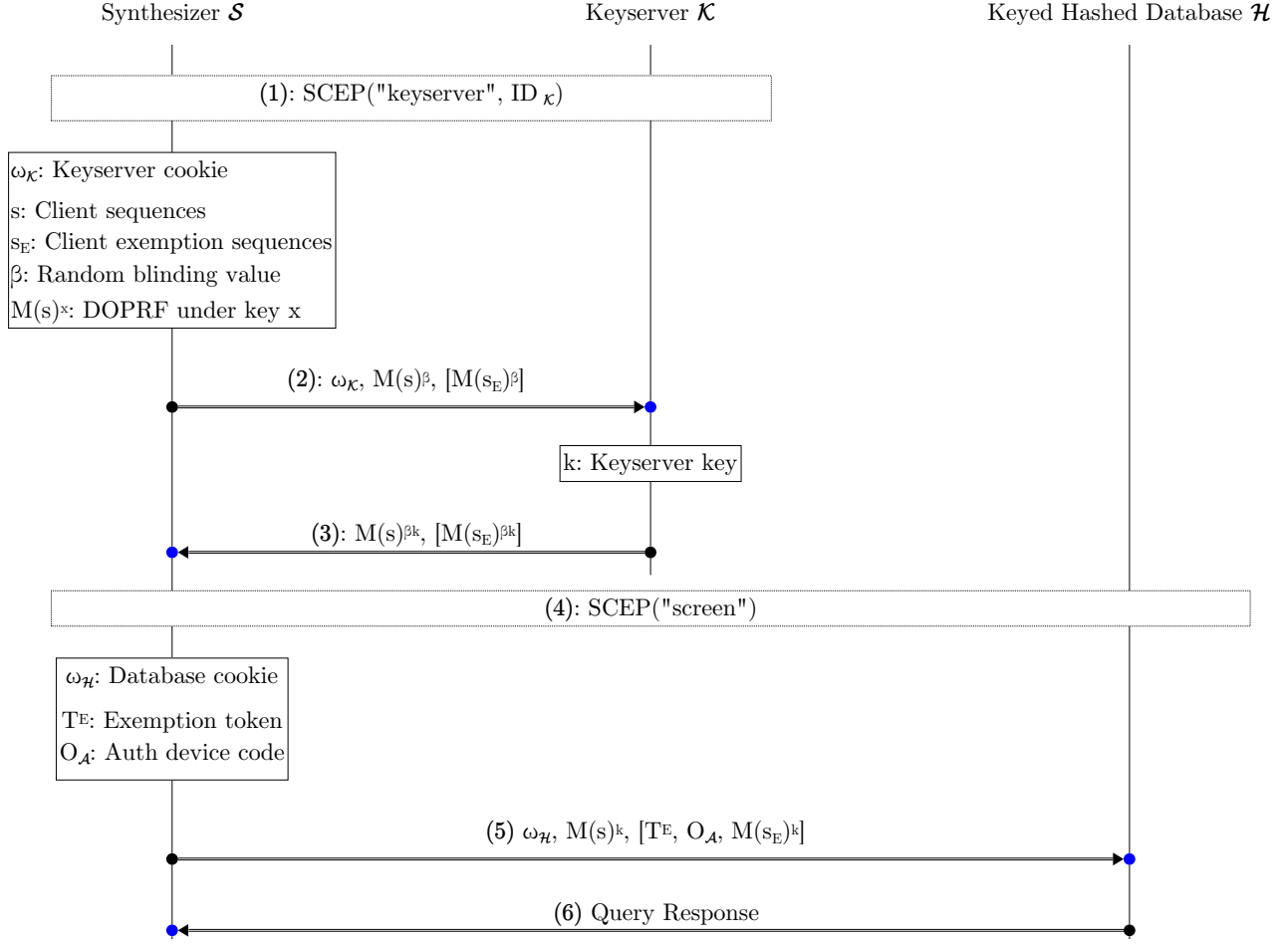
(6) Query Response

Fig. 5. Ladder diagrams of the basic order-request and exemption-handling protocols. Because $\mathcal{S}$ communicates with $\mathcal{K}$ and $\mathcal{H}$ over separate, one-way authenticated TLS connections, $\mathcal{S}$ completes the SCEP mutual authentication protocol (see Figure 4) in Steps 1 and 4 to obtain request cookies $\omega_\mathcal{K}$ and $\omega_\mathcal{H}$. When completing SCEP with $\mathcal{K}$, $\mathcal{S}$ includes $\mathcal{K}$'s identifier $ID_\mathcal{K}$. In Step 2, $\mathcal{S}$ transmits to $\mathcal{K}$ the cookie $\omega_\mathcal{K}$, blinded sequences $M(s)^\beta$ and (optionally) the blinded exempt sequences $M(s_E)^\beta$. In Step 5, $\mathcal{K}$ transmits $\omega_\mathcal{H}$, $M(s)^k$, and an optional tuple (exemption token T^E, second-factor authentication code O_A, and exemption sequence $M(s_E)^k$), to which $\mathcal{K}$ receives a query response that grants or denies the synthesis request in Step 6.