Deniable Key Exchanges for Secure Messaging

Nik Unger
Cheriton School of Computer Science
University of Waterloo,
Waterloo, ON, Canada
njunger@uwaterloo.ca

Ian Goldberg
Cheriton School of Computer Science
University of Waterloo,
Waterloo, ON, Canada
iang@cs.uwaterloo.ca

ABSTRACT

In the wake of recent revelations of mass government surveillance, secure messaging protocols have come under renewed scrutiny. A widespread weakness of existing solutions is the lack of strong deniability properties that allow users to plausibly deny sending messages or participating in conversations if the security of their communications is later compromised. Deniable authenticated key exchanges (DAKEs), the cryptographic protocols responsible for providing deniability in secure messaging applications, cannot currently provide all desirable properties simultaneously.

We introduce two new DAKEs with provable security and deniability properties in the Generalized Universal Composability framework. Our primary contribution is the introduction of Spawn, the first non-interactive DAKE that offers forward secrecy and achieves deniability against both offline and online judges; Spawn can be used to improve the deniability properties of the popular TextSecure secure messaging application. We also introduce an interactive dual-receiver cryptosystem that can improve the performance of the only existing interactive DAKE with competitive security properties. To encourage adoption, we implement and evaluate the performance of our schemes while relying solely on standard-model assumptions.

Categories and Subject Descriptors

K.4.1 [**Computers and Society**]: Public Policy Issues—*Privacy*; C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*

Keywords

Deniability, authenticated key exchanges, secure messaging, universal composability, online repudiation

1. INTRODUCTION

Our society today makes use of large-scale communications platforms such as the Internet, mobile networks, and leased lines to reliably deliver our most critical discourse. However, recent revelations of mass surveillance by intelligence services have highlighted the lack of security and privacy in our messaging tools and spurred demand for better solutions [20]. A widespread weakness in current secure messaging tools is the lack of strong deniability properties [28]. Deniable secure messaging schemes allow conversation participants to later plausibly deny sending messages, or even participating in a conversation, while still providing authentication to the participants at the time of the conversation. This notion was popularized in the secure messaging context with the release of Off-the-Record Messaging (OTR) a decade ago [3]. Unfortunately, the OTR protocol is not well suited to modern settings such as mobile device communication due to its requirement for synchronous connections. Protocol designers seeking to achieve OTR-like deniability properties in these environments have been forced to turn to the cryptographic literature, and have found that existing primitives are not well suited to the task. Different schemes define deniability in slightly different ways, and no existing secure messaging schemes can be said to be deniable under all definitions [28]. Some practitioners have also prematurely dismissed deniability as an impractical property for modern secure messaging applications [15].

Most secure messaging solutions incorporate an authenticated key exchange (AKE) protocol as part of their construction. The goal of an AKE is to establish a fresh shared session key and to authenticate the conversation participants to each other. If the session key cannot be derived from a protocol transcript even when any long-term secret keys are compromised in the future, then the AKE is said to have *forward secrecy*. A deniable authenticated key exchange (DAKE) is an AKE that additionally allows participants to plausibly deny some aspect of their participation in the protocol.

This work has two main goals: to improve the practicality of strong deniability for secure messaging over synchronous connections, and to introduce strong deniability to asynchronous messaging applications such as text messaging. To achieve these goals, we present two new DAKEs that allow deniable message transmission and participation in conversations, even when malicious insiders secretly communicate with judges *during* protocol execution. Among them, we introduce the first non-interactive DAKE achieving both message and participation deniability against these powerful judges while also offering forward secrecy. This DAKE is the first of its kind that can be used in secure messaging applications such as TextSecure [19]. Our DAKEs permit sessions between any parties to be simulated offline with nothing more than the public keys of the alleged participants. We also present a technique that improves the performance of the only existing (3-flow) scheme achieving these levels of deniability. We provide open implementations of our schemes to encourage adoption.

The remainder of this paper is structured as follows: Section 2 discusses the meaning of deniability for secure messaging applications; Section 3 describes the cryptographic primitives that we em-

CCS'15, October 12–16, 2015, Denver, Colorado, USA.

ACM 978-1-4503-3832-5/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2810103.2813616 .

ploy; Section 4 describes an existing scheme with comparable deniability properties, as well as our performance enhancements; Section 5 presents our new DAKE based on ring signatures; Section 6 introduces our one-round DAKE and discusses non-interactive use; Section 7 outlines our implementations, which are evaluated in Section 8; and Section 9 provides our concluding thoughts.

2. DENIABILITY

When we discuss deniability, we must do so with respect to an action and a type of judge. We say that an action is deniable with respect to a given judge if the judge cannot be convinced that an individual performed the action. To make such a statement, we need to define the environment in which the judge resides, and the type of evidence that is required to convince the judge that the action was performed. If an action is deniable with respect to a judge, we say that individuals can "plausibly deny" performing the action. Note that this deniability does not constitute a proof that the parties *did not* perform the action; plausible deniability simply denotes a lack of convincing proof.

There are two primary aspects of conversations that can be called deniable. We can say that messages transmitted during a conversation are deniable (*message repudiation*), but we can also say that participation in the conversation itself is deniable (*participation repudiation*). These properties are orthogonal; a protocol may offer one or the other, both, or neither. For example, messages sent using the well-known OpenPGP protocol are signed with the sender's long-term key, but the signed message does not include the recipient's identity. An OpenPGP-signed email can be used as proof that the message was signed, and presumably authored, by the sender, but not that the sender was in a conversation with the ostensible recipient. Consequently, OpenPGP offers participation repudiation but not message repudiation.

2.1 Judges

When defining a judge, we must define the conditions under which the judge will believe that a user performed a given action, such as sending a message or participating in a conversation. Unfortunately, we must make some assumptions about the behavior of judges if we wish to make meaningful statements about the deniability of secure messaging protocols. If we allow judges to use arbitrary criteria to deliver judgments, then we can never conclude that an action is deniable. In the secure messaging literature, it is common to consider only judges that are completely rational, and decide on the plausibility of an event based solely on the evidence presented to them. The only acceptable evidence for these judges is a valid cryptographic proof, verifiable by the judge, showing that the event must have occurred. In reality, of course, judges are more lenient, and routinely accept plaintext transcripts as evidence. The goal of deniable protocols is to not supply *additional* evidence against a participant, in the form of a hard-to-deny cryptographic proof. Concretely, a messaging protocol that digitally signs every message with the sender's long-term key would not satisfy our notion of deniability, while an unencrypted and unauthenticated protocol would.

Our model of the judge should accurately reflect the quantity and type of evidence required to accept a claim. For example, a digital signature demonstrating that a message was sent either by an accused party or by a specific third party under the condition that this third party had access to unrealistically powerful computational resources may not convincingly provide plausible deniability. If the model of the judge is inaccurate, then we may inadvertently reject deniable schemes or admit non-deniable schemes.

In this paper, we consider only protocols that offer strong deniability. These protocols allow conversation transcripts to be forged by any user with the ability to perform basic computations, while still providing authentication to conversation participants. Consequently, no unforgeable cryptographic proofs can be produced to convince a judge that sessions of these protocols took place.

In addition to the requirements for evidence, we must also define the judge's relationship to the protocol participants and the judge's capabilities. Two primary types of judges have been discussed in the secure messaging literature: offline judges, and online judges.

An **offline judge** examines the transcript of a protocol execution that occurred in the past, and decides whether or not the event in question occurred. A judge of this type is given a *protocol transcript*, showing all of the (usually encrypted) transmitted data, and a *chat transcript*, showing the high-level chat messages that were exchanged. The judge must then decide whether the protocol and chat transcript constitute proof that the action in question occurred. When proving the deniability of protocols, it is also normally assumed that an offline judge is given access to the long-term secrets of all parties named in the transcript. Our goal is to prevent judges from distinguishing between real and fake transcripts.

An **online judge** interacts with a protocol participant, referred to as the *informant*, while the secure messaging protocol is being executed. The judge has a secure and private connection to the informant, and may instruct the informant to perform actions in the protocol. The goal of the judge is to evaluate whether the actions of other participants in the protocol are actually occurring, or if the informant is fabricating the conversation (i.e., they are a *misinformant*). The judge does not have any direct visibility into the network, but it may instruct the informant to corrupt participants, compromising their secret keys. The judge is notified when a participant is corrupted.

2.2 Practicality of Deniability

The secure messaging community has previously debated whether or not deniability should be implemented in end-user tools [15]. There are two main arguments against designing deniable messaging protocols: deniability properties are too expensive to implement, and these properties are not useful in practice. However, the relevant deniability properties for secure messaging protocols can now be implemented relatively inexpensively and without any usability consequences [28].

While the secure messaging literature mostly focuses on judges that understand cryptography and rely on cryptographic proofs to make decisions, real-world judges often do not behave in this manner, and routinely accept plaintext transcripts. We cannot design protocols that provide more deniability than plaintext; however, we can easily design protocols that provide *less* deniability—while a real-world judge of this type may not accept arguments that a plaintext transcript could theoretically be forged, they may be likely to accept testimony from experts that a protocol containing a digital signature could *not* be forged. For these reasons, we should strive to design deniable protocols to avoid unintentionally incriminating users.

3. CRYPTOGRAPHIC PRIMITIVES

We make use of several specialized cryptosystems when constructing our schemes. This section provides high-level definitions of these underlying cryptosystems and outlines our notation.

We write $r \xleftarrow{\$} S$ to denote that r is assigned an element from set S selected uniformly at random. For all schemes, we implicitly assume that a security parameter λ is provided to control the security level of the system. We abuse notation by assuming that

$r \xleftarrow{\$} \{0,1\}^\lambda$ produces a random binary value with an appropriate length for the task at hand, with a security level controlled by λ. Several functions we define accept a parameter r that denotes the randomness associated with operations; if omitted, $r \xleftarrow{\$} \{0,1\}^\lambda$ is used. We denote concatenation of values with the $\|$ operator. We implicitly assume that all concatenated values are of a fixed width.

We denote a traditional digital signature scheme as three functions: **SigGen**(r), a key generation function producing key pair (pk, sk); **Sig**(pk, sk, m, r), a signing function producing a signature σ of message m; and **Vrf**(pk, σ, m), a verification function. Our only security requirements for digital signature schemes are correctness, soundness, and consistency [5]. Similarly, we denote a public-key cryptosystem as: **PKGen**(r), **PKEnc**(pk, m, r), and **PKDec**(pk, sk, γ), where γ is the ciphertext. Our constructions that make use of public-key cryptosystems require that they are IND-CCA2 secure [22].

Dual-Receiver Encryption (DRE). It is sometimes desirable to encrypt a message such that it can only be read by either of two named recipients, and that anyone can verify this fact. Dual-receiver encryption (DRE) is a type of specialized cryptosystem that enables publicly verifiable encryptions of messages for two receivers. A DRE scheme consists of three functions: **DRGen**(r), a key generation function; **DREnc**(pk_1, pk_2, m, r), an encryption function; and **DRDec**$(pk_1, pk_2, sk_i, \gamma)$, a decryption function for $i \in \{1, 2\}$. DREnc encrypts a message m under two public keys: pk_1 and pk_2. To achieve correctness, DRE schemes must satisfy DRDec$(pk_1, pk_2, sk_i, \text{DREnc}(pk_1, pk_2, m, r)) = m$ for any (pk_1, sk_1) and (pk_2, sk_2) produced by DRGen, $i \in \{1, 2\}$, and any m and r. We require DRE schemes to exhibit symmetry, public verifiability, completeness, soundness, and dual-receiver IND-CCA1 security, as defined by Chow et al. [9].

Non-Committing Encryption (NCE). A public-key cryptosystem is called *non-committing* if, in addition to offering the standard functions of a PKE scheme, it can also produce "rigged" ciphertexts [6]. A non-committing cryptosystem consists of five functions: **NCGen**, **NCEnc**, and **NCDec**, which behave identically to the corresponding functions of a PKE scheme; **NCSim**(r), a rigged ciphertext generation function; and **NCEqv**(pk, γ, α, m), an equivocation function. NCSim produces: pk, a public key; γ, a ciphertext; and α, some auxiliary information. Public keys and ciphertexts produced by NCSim are identically distributed to public keys and ciphertexts produced by NCGen and NCEnc, respectively. If NCEqv is called with (pk, γ, α) produced by NCSim, it generates values sk, r^*, and r^{NCE} such that NCGen$(r^*) = (pk, sk)$ and NCEnc$(pk, m, r^{NCE}) = \gamma$. In other words, it uses α to generate a secret key sk corresponding to pk such that the previously generated γ decrypts to m using sk. All of this is accomplished while making key pair (pk, sk) appear as though it were honestly generated using NCGen, and γ as though it was honestly generated using NCEnc. There are several existing NCE constructions [6,8,11,31], but they are all considerably more expensive than the other primitives used in our protocols.

Ring Signatures. Ring signature schemes, originally proposed by Rivest et al. [24], are a specialized type of digital signature scheme. A ring signature is verifiably produced by a member of a given set, but the exact identity of the signer cannot be determined. A ring signature scheme consists of three functions: **RSGen**(r), a key generation function; **RSig**(pk, sk, R, m, r), a signing function; and **RVrf**(R, σ, m), a verification function. The ring R is a set of n public keys $\{pk_1, pk_2, \ldots, pk_n\}$ that could possibly have produced the signature. It is required that $n > 1$, (pk, sk) was generated with RSGen, and $pk \in R$. r controls the randomization of the output. We require the use of ring signature schemes exhibiting correctness, anonymity against full key exposure, and unforgeability with respect to insider corruption [2].

4. THE WALFISH PROTOCOL

In his thesis, Walfish [29] introduced Φ_{dre}.[1] Φ_{dre} is the only previously defined DAKE of which we are aware that provides deniability against both online and offline judges while simultaneously providing forward secrecy.

To prove the security of Φ_{dre}, Walfish extended Canetti's Universal Composability (UC) framework [4] to produce the Generalized UC (GUC) framework [29]. Security proofs in the UC framework demonstrate that real protocols operating in the presence of an adversary $\mathcal{A}$ behave identically to idealized protocols, wherein a trusted central server with secure connections to the protocol participants executes an "ideal functionality" program, in the presence of an adversary $\mathcal{S}$; in other words, these settings are indistinguishable from the perspective of a "distinguishing environment" $\mathcal{Z}$. A common operation for ideal functionalities is to send a "delayed message" to a party, where the adversary $\mathcal{S}$ is allowed to withhold or delay the message arbitrarily (even if the message contents are hidden from $\mathcal{S}$). Unfortunately, the UC framework does not provide security guarantees when information is shared between protocol sessions (e.g., as in a PKI). GUC differs from UC in that it allows multiple concurrent protocol sessions with shared information that is accessible by both the adversary and $\mathcal{Z}$. Additionally, $\mathcal{Z}$ is permitted to execute arbitrary other protocols. Shared information is represented by persistent machines running "shared functionalities". A GUC-based security model in which a shared functionality $\mathcal{G}$ is accessible is called a $\mathcal{G}$-hybrid model. Despite this additional distinguishing power, GUC can still provide the usual UC security guarantees. A simplified but equivalent framework, the External-subroutine UC (EUC) framework, can be used to prove security in the GUC framework more easily [29]. Throughout the remainder of this work, we assume some familiarity with the UC framework.

4.1 Protocol Φ_{dre}

Φ_{dre}, depicted in Figure 1, is a two-round interactive DAKE.[2] Each message sent between I and R (other than the introductory message asserting identities) is encrypted using DRE under the public keys of I and R. This guarantees that all three messages can be read by both I and R, but nobody else (unless either I or R has been corrupted). To provide authentication, the core of the protocol involves each party echoing a nonce generated by their partner, thereby proving that they can decrypt the communications. Since only I and R can decrypt the DRE, and each party knows that they did not produce the response to their nonce themselves, this provides non-transferable authentication. In addition, I includes in its encrypted message an ephemeral public key pk for an NCE scheme. R generates the shared secret for the session, and encrypts it using pk as part of its last message to I.

It is trivial for anyone with access to the long-term public keys to forge protocol transcripts between any two parties, even without access to any long-term secret keys. An offline forger of this type merely needs to simulate both participants; although it cannot decrypt the DRE layer itself, it already knows the contents of all

[1] Φ_{dre} was later restated in a publication by Dodis et al. [12].

[2] In practice, the protocol can be collapsed into three flows by having the party denoted as R in Figure 1 send the session identifier and party names.

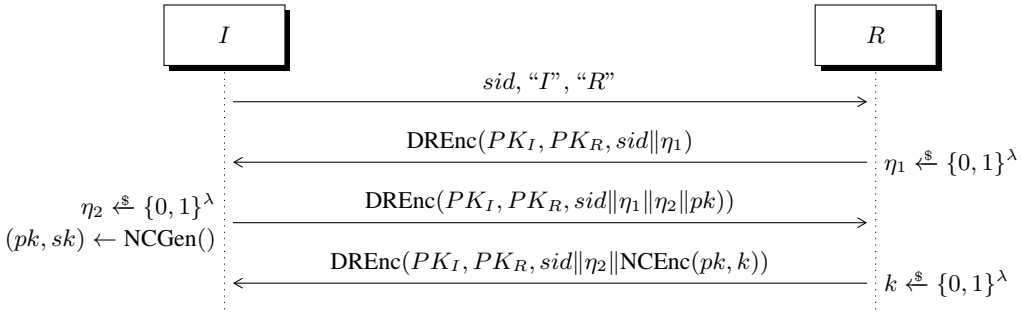

Figure 1: Φ_{dre} [29]. The shared secret is k.

messages. Deniability against online judges is provided because each party can simulate the behavior of the other; the DRE allows misinformants to read the contents of any messages that the judge produces on their behalf. The use of NCE is only needed in the non-erasure model (where we assume that memory cannot be erased) with semi-adaptive corruptions (i.e., the adversary may not corrupt participants while the protocol is executing); if erasures are allowed or corruptions are only static, then a standard PKE scheme suffices.

Walfish used the GUC framework to prove that Φ_{dre} provides its claimed security and deniability properties with semi-adaptive corruptions in the $\bar{\mathcal{G}}_{krk}$-hybrid model [29]. $\bar{\mathcal{G}}_{krk}$ (key registration with knowledge) is a shared functionality that models a PKI. Any party P can register for a key pair with $\bar{\mathcal{G}}_{krk}$. $\bar{\mathcal{G}}_{krk}$ generates the requested key pair directly, thereby modeling a PKI that requires proof of knowledge of secret keys. A corrupted party can override this by providing its own key pair for storage. PK_P, the public key for a party P, can be retrieved from $\bar{\mathcal{G}}_{krk}$ by any machine. $\bar{\mathcal{G}}^{\mathcal{F}}_{krk}$ also allows SK_P, the secret key for party P, to be retrieved by an honest machine executing $\mathcal{F}$ or by P if it has been corrupted.

The deniability of Φ_{dre} is necessarily imperfect: if an adversary is willing to disrupt the protocol in such a way that it is guaranteed to abort before completion, then incriminating information demonstrating that I was attempting to communicate with R can be released. If Justin, an online judge, instructs Mallory, the adversary, to modify the first flow from R to I to an encryption of nonce η_1' known only to Justin (and thereby inevitably cause R to abort), then Mallory will need to provide I's response to R. This requires Mallory to actually interact with the real I (i.e., Mallory is a legitimate informant), or to compromise SK_I or SK_R without the knowledge of Justin (something disallowed by the GUC framework). If Mallory attempts to simulate a response, then Justin can later instruct Mallory to corrupt a party to recover their secret key. Justin can then use this secret key to decrypt the simulated response, and determine that it did not include η_1'. Walfish accounts for this weakness of Φ_{dre} in the security proof by defining an *incrimination procedure* called IncProc. If the adversary causes the protocol to abort, it gains access to IncProc, which releases incriminating information that precisely models the real-world leakage.

4.2 Efficient Standard-Model Instantiation

In order to implement Φ_{dre}, we must select a DRE scheme to use. Unfortunately, nearly all existing DRE schemes require use of the random oracle model. The DRE construction of Chow et al. [9] is relatively efficient and is secure in the standard model, but it is still expensive compared to simple encryption schemes. Alongside the original definition of Φ_{dre}, Walfish describes a construction of a generic DRE scheme [29]. This scheme involves encrypting the plaintext twice using a PKE scheme with IND-CCA2 security, and then providing two non-interactive zero-knowledge proofs of knowledge (NIZKPK) demonstrating that the encrypted plaintexts are equal. Unfortunately, NIZKPK schemes are either highly inefficient or require random oracles. In this section, we describe a new DRE construction that improves the performance of Φ_{dre} while still maintaining its security properties in the standard model.

We begin by making an important observation about Φ_{dre}: it is an interactive protocol that takes place between two known parties. While DRE in general is a non-interactive protocol, allowing the DRE scheme to require interactivity does not negatively impact the usability of the overall scheme. We are able to do this because for each encryption of a message, the only party that will need to decrypt the message is available for interactive communications.

Our basic approach to the construction is similar to Walfish's general DRE scheme, but we use an interactive zero-knowledge proof of knowledge (ZKPK) scheme instead. While we will only describe one possible instantiation, any PKE scheme with IND-CCA2 security can be combined with any interactive ZKPK of plaintext equality. This DRE remains "publicly verifiable" in the sense that the ZKPK verifier can verify the correctness of the ciphertexts even if they do not know any secret keys; this is sufficient for use in Φ_{dre}. As a PKE scheme, we make use of the cryptosystem published by Cramer and Shoup [10]. The Cramer-Shoup scheme provides IND-CCA2 security in the standard model with only the DDH assumption. To prove that the two ciphertexts contain identical messages and are of a valid format, we use a Σ ZKPK of the kind described by Schnorr [25]. The resulting scheme is secure with only the DDH assumption for the underlying Cramer-Shoup group, and consists of the following functions:

DRGen(r): key generation is the same as in the Cramer-Shoup scheme [10]. The resulting public key for a user consists of a group description (G, q, g_1, g_2) and values $c = g_1^{x_1} g_2^{x_2}$, $d = g_1^{y_1}$, and $h = g_1^z$. The corresponding secret key is (x_1, x_2, y_1, y_2, z).

DREnc(pk_1, pk_2, m, r): m is encrypted twice using the Cramer-Shoup scheme (once for each public key), and a ZKPK of plaintext equality is produced. r is interpreted as $r = k_1 \| k_2$ and is used to randomize the encryptions of m. Given public key $pk_i = (G_i, q_i, g_{1i}, g_{2i}, c_i, d_i, h_i)$, the ciphertexts consist of $u_{1i} = g_{1i}^{k_i}$, $u_{2i} = g_{2i}^{k_i}$, $e_i = h_i^{k_i} m$, and $v_i = c_i^{k_i} d_i^{k_i \alpha_i}$ for $i \in \{1, 2\}$ and $\alpha_i = H(u_{1i} \| u_{2i} \| e_i)$ using a collision-resistant hash function H.

The result also includes an interactive ZKPK that proceeds between the prover $\mathcal{P}$ (the party calling DREnc) and the verifier $\mathcal{V}$ (the party that will call DRDec) as follows:

1. $\mathcal{P}$ generates random values $m_i \in [0, q_i)$ for $i \in 1, 2$. $\mathcal{P}$ then computes $T_{1i} = g_{1i}^{m_i}$, $T_{2i} = g_{2i}^{m_i}$, $T_{3i} = (c_i d_i^{\alpha_i})^{m_i}$, and $T_4 = \frac{h_1^{m_1}}{h_2^{m_2}}$, and sends these values to $\mathcal{V}$.

2. $\mathcal{V}$ generates random value L and sends it to $\mathcal{P}$.

3. $\mathcal{P}$ computes $n_i = m_i - Lk_i \pmod{q_i}$ for $i \in \{1, 2\}$ and sends these values to $\mathcal{V}$.

4. $\mathcal{V}$ accepts the encryption as valid if the following equalities hold for $i \in \{1, 2\}$: $T_{1i} \stackrel{?}{=} g_{1i}^{n_i} u_{1i}^L$, $T_{2i} \stackrel{?}{=} g_{2i}^{n_i} u_{2i}^L$, $T_{3i} \stackrel{?}{=} (c_i d_i^{\alpha_i})^{n_i} v_i^L$, and $T_4 \stackrel{?}{=} \frac{h_1^{n1}}{h_2^{n2}} (\frac{e_1}{e_2})^L$.

DRDec$(pk_1, pk_2, sk_i, \gamma)$: γ is parsed to locate the encryption for pk_i, and decryption proceeds as in the Cramer-Shoup scheme. Let $sk_i = (x_{1i}, x_{2i}, y_{1i}, y_{2i}, z_i)$. At this point, the recipient of γ has already verified that the ciphertexts are of the correct form and that they contain encryptions of the same message as a result of the interactive ZKPK. In addition, the recipient computes $\alpha_i = H(u_{1i} \| u_{2i} \| e_i)$ and then verifies that $u_{1i}^{x_{1i}} u_{2i}^{x_{2i}} (u_{1i}^{y_{1i}} u_{2i}^{y_{2i}})^{\alpha_i} \stackrel{?}{=} v_i$. The message m is recovered using $m = \frac{e_i}{u_{1i}^{z_i}}$.

The resulting protocol consists of 9 messages (plus an additional message for the introductory identity assertions). This instantiation of Φ_{dre} is very efficient compared to implementations using non-interactive DRE in the standard model, which typically require hundreds of group elements to be transmitted [9]. While we do not prove the security of this variant here, the original proof by Walfish [29] can be extended without issue, as this interactive DRE construction satisfies all of the required properties of the original protocol definition. We caution that the interactive ZKPK sessions must not be interleaved in order to preserve online repudiation (i.e., the verifier must wait for all ZKPK message flows to complete before decrypting the message). We also note that, if an implementer is willing to accept the use of the random oracle model, then this instantiation of DRE can be made non-interactive through the use of the Fiat-Shamir heuristic [13].

5. AN EFFICIENT DAKE SCHEME FROM RING SIGNATURES

While Φ_{dre} can be made efficient through the use of interactive DRE, the resulting protocol requires 9 flows to complete the key exchange. In environments with high latency, such an approach may be undesirable. Additionally, Φ_{dre} is a *non-contributory* key exchange; the resulting shared secret is chosen entirely by a single party (the responder R). Consequently, an adversary that has corrupted R can cause the initiator I to use an adversarially chosen key. Moreover, while Φ_{dre} allows parties to transmit identities within the protocol, the ideal functionality used by Walfish to prove the scheme's security represents a *pre-specified peer* key exchange (i.e., both parties must know the identity of the other participant before the protocol begins), which fails to capture this feature.

All of these limitations can be overcome by a family of key exchanges known as SIGMA ("SIGn-and-MAc") protocols. First proposed by Krawczyk [16], SIGMA protocols are *contributory* (both parties ensure the randomness and freshness of the resulting key), consist of only 3 message flows, and permit *post-specified peers* (i.e., the identity of the other party is an output of the protocol). Canetti and Krawczyk have previously shown that a basic SIGMA protocol is UC-secure in the $\mathcal{F}_{SIG}$-hybrid model with adaptive corruptions [7]. Unfortunately, this proof shares the limitations of all proofs in plain UC-based models, including a failure to model public key directories that are available to the distinguishing environment. Additionally, no SIGMA protocols can match the strong deniability properties offered by Φ_{dre}.

In this section, we make use of ring signatures to construct a new deniable key exchange protocol, inspired by SIGMA designs, that offers provably strong security and deniability in the GUC framework. The resulting protocol, RSDAKE, is not a true SIGMA protocol (since it does not need to use a MAC), but it addresses all of the aforementioned problems with Φ_{dre}.

Algorithm 1 Ideal functionality $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$

on receipt of $(\texttt{initiate}, sid, I, SK_I)$ **from** I:
  if (I is "active") **return**
  Mark I as "active"
  Send $(\texttt{initiate}, sid, I)$ to $\mathcal{S}$
on receipt of $(\texttt{establish}, sid, R, SK_R)$ **from** R:
  if (I is not "active") **return**
  if ((R is "active") || (R is "aborted")) **return**
  Mark R as "active"
  Send $(\texttt{establish}, sid, R)$ to $\mathcal{S}$
  $k \stackrel{\$}{\leftarrow} \{0,1\}^\lambda$
on receipt of $(\texttt{set-key}, sid, T, P', k')$ **from** $\mathcal{S}$:
  if (a `set-key` message was already sent to T) **return**
  if (($T \notin \{I, R\}$) || (T is not "active")) **return**
  Let $P \in \{I, R\}$ such that $P \neq T$
  if (($P' \neq P$) **&&** (P' is uncorrupted)) **return**
  if ((I is corrupt) || (R is corrupt)) {
    Send $(\texttt{set-key}, sid, P', k')$ to T
  } **else** {
    Send $(\texttt{set-key}, sid, P, k)$ to T
  }
  if (two `set-key` messages have been sent) Halt
on receipt of $(\texttt{abort}, sid, I, R)$ **from** $\mathcal{S}$:
  if (I is "active") Send delayed $(\texttt{abort}, sid, I)$ to I
  if (R is "active") {
    Mark R as "aborted"
    Send delayed $(\texttt{abort}, sid, R)$ to R
  }
on receipt of $(\texttt{incriminate}, sid, I, R)$ **from** $\mathcal{S}$:
  if (already received `incriminate` message) **return**
  if ((R is "aborted" and honest) **&&** (I is "active")) {
    Execute IncProc$(sid, I, R, PK_I, PK_R, SK_R, k)$
  }

5.1 Functionality $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$

Before defining RSDAKE, we begin by formulating an ideal functionality in the GUC framework that captures the desired properties. The new functionality, $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ (post-specified peer key exchange with incriminating abort), is given in Algorithm 1. The name and operation of $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ is partially based on the $\mathcal{F}_{keia}^{\text{IncProc}}$ functionality defined by Walfish, which was used to prove the security of Φ_{dre} [29]. In the remainder of this section, we discuss the behavior of $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ and the design decisions behind it.

To prove the security of SIGMA protocols using the UC framework, Canetti and Krawczyk defined $\mathcal{F}_{post\text{-}ke}$, an ideal functionality that models a key exchange with post-specified peers [7]. Like $\mathcal{F}_{post\text{-}ke}$, $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ takes place between an unbounded number of parties, but each session captures the interaction between only two of these parties. The first party to request a key exchange is subsequently known as I, the initiator. The second party to request a key exchange is subsequently known as R, the responder. After both I and R are known, $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ selects a random shared key k for the session. The adversary is then given a chance to attempt to set the output (the shared key and the identity of the other party) of both I and R. If the adversary has corrupted either party, then

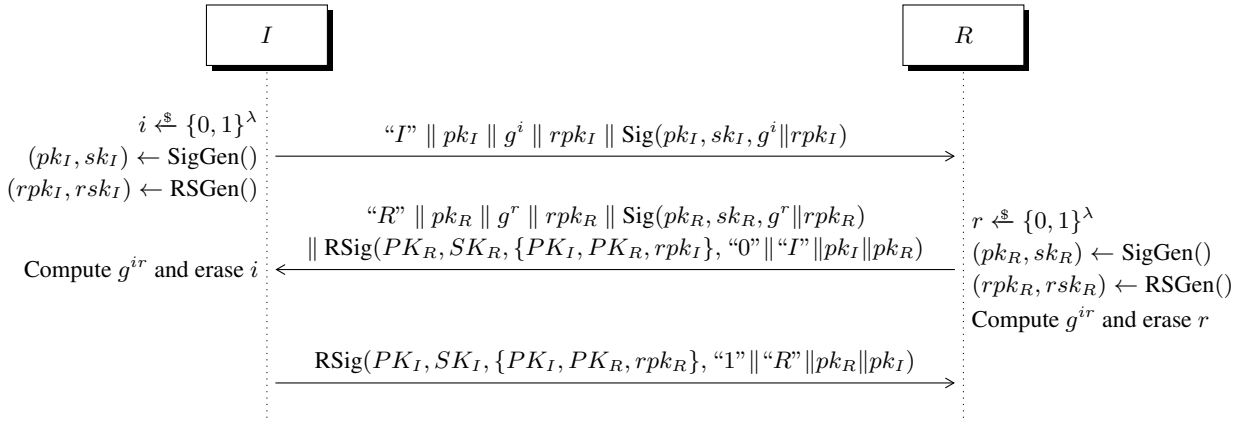

Figure 2: RSDAKE. The shared secret is g^{ir}.

it is given the ability to send an adversarially chosen secret k' and partner identity P to I and R.[3] Otherwise, each party is given k and the true identity of their conversation partner.

Unfortunately, $\mathcal{F}_{post\text{-}ke}$ cannot be realized in the GUC framework in the $\bar{\mathcal{G}}_{krk}$-hybrid model. Similarly to $\mathcal{F}_{keia}^{\text{IncProc}}$, $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ must explicitly weaken the deniability of the protocol by allowing for incriminating aborts. Concretely, we allow the adversary to abort the protocol in order to cause R to output incriminating information. The nature of this incriminating information is a parameter to $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ in the form of a procedure IncProc, allowing it to be tailored to the real protocol under consideration. When the adversary $\mathcal{S}$ asks $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ to abort, an instance of IncProc is started. $\mathcal{S}$ cannot cause an uncorrupted R to output a key after the protocol has aborted, but it may still cause I to output a key by withholding the `abort` message sent to I.

There are several subtle differences between the $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ and $\mathcal{F}_{keia}^{\text{IncProc}}$ functionalities. In $\mathcal{F}_{keia}^{\text{IncProc}}$, it is the initiator I that releases incriminating information when the protocol aborts. In contrast, if the RSDAKE exchange aborts, incriminating information will be released by the responder R. Additionally, since incriminating aborts in $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ occur after R has generated k, IncProc must also accept k as input.

5.2 RSDAKE

Our new protocol, RSDAKE, is presented in Figure 2. Each protocol participant P has a long-term key pair (PK_P, SK_P) for a ring signature scheme, where PK_P is publicly known. To begin a session, the initiator I generates an ephemeral signing key pair (pk_I, sk_I), an ephemeral Diffie-Hellman public key g^i, and a ring signature key pair (rpk_I, rsk_I). It sends its identity, its ephemeral public keys, and a signature of $g^i \| rpk_I$ using pk_I to R. This signature binds the ephemeral keys for the different schemes to the same "master" key pk_I. This first message is referred to as ψ_1.

Responder R performs the same procedure and responds with pk_R, g^r, and rpk_R. It also performs a ring signature of the two ephemeral master keys pk_I and pk_R as well as the identity of I. The response message is referred to as ψ_2. The ring used for this signature is $\{PK_I, PK_R, rpk_I\}$. This ring signature serves the same purpose as the (traditional) signature and MAC in the basic SIGMA protocol. An honest I is convinced that R produced the signature because it knows that no other parties have access to SK_I or rsk_I. It also knows that this signature has not been reused from another session (because it contains pk_I and pk_R), and that R believes that it is communicating with the correct partner (because the signature contains the identity of I). However, this proof is not transferable to any other party because the signature could have also been forged by I using SK_I.

In the third and final step of the protocol, I responds with its own ring signature of the master ephemeral keys and the identity of R, computed over the ring $\{PK_I, PK_R, rpk_R\}$. This final message is referred to as ψ_3. R is convinced of I's identity, but cannot transfer this conviction, for the same reasons as before. The resulting shared secret is g^{ir}, as in a standard Diffie-Hellman exchange.

Unlike SIGMA protocols, RSDAKE offers offline repudiation equal to that of Φ_{dre}. Specifically, anyone can forge a key exchange (and subsequent conversation) between any two parties I and R using nothing other than PK_I and PK_R. An offline forger is in the unique position of generating ephemeral keys for both simulated parties, and so it can compute both ring signatures using rsk_I and rsk_R. Transcripts generated by such a forger are indistinguishable from real transcripts due to the security of the ring signature scheme. If the ring signature scheme provides security under full-key exposure, this indistinguishability holds even if the long-term secret keys of both I and R are subsequently compromised.

Assuming the existence of a signature scheme and a ring signature scheme that is secure under full-key exposure, RSDAKE GUC-realizes $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ within the erasure $\bar{\mathcal{G}}_{krk}^{\text{RSDAKE}}$-hybrid model with fully adaptive corruptions. For brevity, we omit proof of this claim here; a formal proof can be found in the first author's Master's thesis [27, §3.7.3].

6. A NON-INTERACTIVE DAKE

Both Φ_{dre} and RSDAKE have a usability limitation: they are interactive protocols (i.e., they require both parties to be online). In applications such as secure messaging, the key exchange must be completed before messages can be transmitted. In some domains, such as instant messaging, consistent peer availability may be a valid assumption. However, email and text messaging are two extremely popular systems in which interactive key exchanges cannot be used in general. These environments benefit from the use of non-interactive key exchanges; secure messages can be sent immediately to any peer in the network, irrespective of their connectivity.

TextSecure [19] is a popular secure messaging application for mobile phones that offers one of the most comprehensive sets of

[3] $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ models a scenario in which the adversary $\mathcal{S}$ can completely control the value of the shared secret key after corrupting only one party. In a contributory key exchange, $\mathcal{S}$ may not have full control over this value, but it can still influence the result.

Algorithm 2 Ideal functionality $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$

on receipt of $(\texttt{solicit}, sid, I, SK_I)$ **from** I:
  if (I is "active") || (I is "aborted")) **return**
  Mark I as "active"; record $(\texttt{initiator}, sid, I, SK_I)$
  Send $(\texttt{solicit}, sid, I)$ to $\mathcal{S}$

on receipt of $(\texttt{establish}, sid, I, R, SK_R)$ **from** R:
  if (($\texttt{solicit}$ not received) || (R is "active")) **return**
  Mark R as "active"; record $(\texttt{responder}, sid, R, SK_R)$
  Send $(\texttt{establish}, sid, I, R)$ to $\mathcal{S}$
  $k \xleftarrow{\$} \{0,1\}^\lambda$

on receipt of $(\texttt{set-key}, sid, P, k')$ **from** $\mathcal{S}$:
  if ((k is set) **&&** ($P = R$ || P is corrupt)) {
    if (R is corrupt) {
      Send $(\texttt{set-key}, sid, I, P, k')$ to R
      if (I is "active") {
        Send delayed $(\texttt{set-key}, sid, I, P, k')$ to I
      }
    } **else** {
      Send $(\texttt{set-key}, sid, I, R, k)$ to R
      if (I is "active") {
        Send delayed $(\texttt{set-key}, sid, I, R, k)$ to I
      }
    }
  }
  Halt

on receipt of $(\texttt{abort}, sid, I, R)$ **from** $\mathcal{S}$:
  if (I is "active") {
    Mark I as "aborted"
    Send delayed $(\texttt{abort}, sid, I, R)$ to I
  }

on receipt of $(\texttt{incriminate}, sid, R)$ **from** $\mathcal{S}$:
  if (already received $\texttt{incriminate}$ message) **return**
  if ((I is "aborted") **&&** (R is "active") **&&** (R is honest)) {
    Execute IncProc$(sid, I, R, PK_I, PK_R, SK_R, k)$
  }

security features [28]. TextSecure currently uses the triple Diffie-Hellman (3-DH) DAKE, but this protocol does not offer online repudiation. In this section, we present a secure and deniable one-round key exchange protocol that can be used in interactive or non-interactive settings, with the ultimate goal of improving the deniability of TextSecure.

6.1 Functionality $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$

To prove that our key exchange is secure in the GUC framework, we must define an ideal functionality that captures the behavior of TextSecure. Unfortunately, neither $\mathcal{F}_{keia}^{\text{IncProc}}$ nor $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ fully describe the desired properties.

In TextSecure, the initiator I begins by uploading "prekeys" to a central server. A prekey is a set of ephemeral information that can be used as the first flow of a one-round key exchange protocol. Subsequently, the responder R requests the next available prekey for I and uses it to complete the key exchange. In practice, the first message of the conversation is encrypted under this key and attached to the same flow. Even if I is offline when this message is sent, and R then goes offline forever, I will still be able to decrypt this message when it comes back online. It is important to note that I does not know the identity of the party that will respond to the prekeys it produces, but R knows the identity of the party to whom it wishes to send a message. In this sense, the key exchange has a *single post-specified peer*. Concretely, the identity of R should be part of the *output* for I, while the identity of I is an *input* for R. Neither $\mathcal{F}_{keia}^{\text{IncProc}}$ nor $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$ captures this notion.

Similarly to $\mathcal{F}_{post\text{-}keia}^{\text{IncProc}}$, information generated by IncProc incriminates the responder R, and IncProc is called after the shared secret k is generated. We define ideal functionality $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ (single post-specified peer key exchange with incriminating abort) to capture the desired properties. $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ takes place between two or more parties. The functionality is given in Algorithm 2.

In the normal case, a party informs $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ that it would like to solicit a connection. Without loss of generality, we call this party I. Only one party solicits a connection per session. Later, another party asks the functionality to complete an exchange with I. Without loss of generality, we call this party R. $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ generates a shared secret k, and then waits for the adversary $\mathcal{S}$ to issue a $\texttt{set-key}$ request. If $\mathcal{S}$ has not corrupted R, then $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ instantly sends k to R and then sends k and the identity of R as a delayed message to I.

$\mathcal{S}$ can choose to withhold the final message sent to I, causing R to complete and I to stall. If $\mathcal{S}$ corrupts R before the protocol halts, then it can cause I to receive an adversarially chosen secret k' and the identity of any corrupted party P. Finally, $\mathcal{S}$ is also allowed to abort the protocol after I has solicited a connection, and it can choose whether or not I should be informed of this abort. In any case, if the protocol is aborted, $\mathcal{S}$ can cause R to generate incriminating information that proves R was attempting to communicate with I. $\mathcal{S}$ cannot cause I to output a key if the protocol was aborted, but it can still cause R to output a key.

$\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ is parameterized with an incriminating abort procedure IncProc that accepts the as parameters the session identifier, the identities of I and R, the long-term public keys of I and R, the secret key of R, and the shared secret k.

In practice, at most one honest respondent replies to each initial message from I. If dishonest parties send multiple responses to I, then I processes only the first response received. $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ captures this situation by binding each instance of the functionality (i.e., each protocol "session") to a distinct $\texttt{solicit}$ message sent by I. Therefore, at most one party is labeled as "R" for each session; subsequent $\texttt{establish}$ messages to $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ are ignored. The GUC framework spawns a new instance of $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ for each session within the network, while still supporting shared state between them through the use of shared functionalities.

6.2 Spawn*

We now define a one-round DAKE, Spawn*, that can be used in interactive or non-interactive settings. Spawn* provides both offline and online message and participation repudiation (with limited exceptions that we discuss later) while also providing forward secrecy for completed sessions.[4] Figure 3 depicts the protocol, which takes place between an initiator I and a responder R.

Before any sessions begin, all parties register long-term public keys with a PKI. Each party P generates a master signing keypair $(PK_P^{Sig}, SK_P^{Sig}) \leftarrow \text{SigGen}()$, and two scheme-specific keypairs: $(PK_P^{DRE}, SK_P^{DRE}) \leftarrow \text{DRGen}()$ as well as $(PK_P^{RS}, SK_P^{RS}) \leftarrow \text{RSGen}()$. P binds the keys together by computing a signature $\sigma_P^{PKI} \leftarrow \text{Sig}(PK_P^{Sig}, SK_P^{Sig}, PK_P^{DRE} \| PK_P^{RS})$ and uploading $PK_P = (PK_P^{Sig}, PK_P^{DRE}, PK_P^{RS}, \sigma_P^{PKI})$ to the PKI along with proofs of knowledge of the corresponding secret keys. When retrieving PK_P from the PKI, parties verify the trustworthiness of

[4] In a well-known result, Bellare et al. have previously shown that no one-round protocol can achieve the strongest notion of forward secrecy [1]. Here, we discuss "weak forward secrecy".

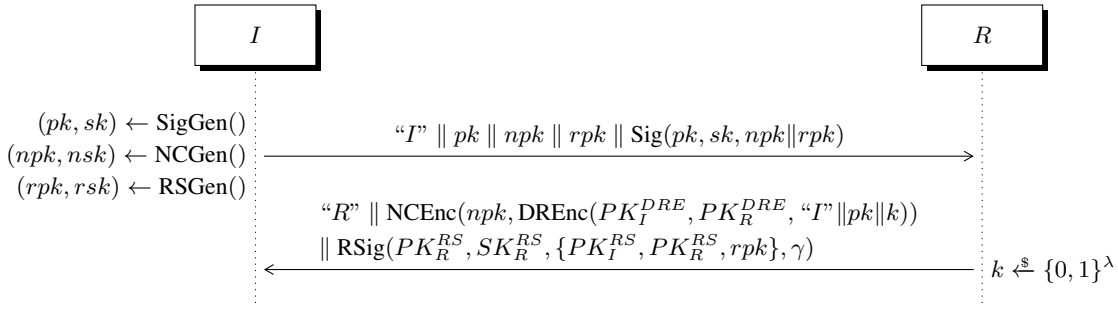

Figure 3: Spawn*. The shared secret is k. γ denotes "R" concatenated with the output of NCEnc. Spawn replaces NCGen with PKGen and NCEnc with PKEnc. In all other ways, Spawn is identical to Spawn*.

PK_P^{Sig} using a scheme-specified key verification mechanism. This trust is extended to PK_P^{DRE} and PK_P^{RS} by verifying σ_P^{PKI}.

When starting a session, I uses a standard signature scheme to generate an ephemeral key pair (pk, sk). This key pair is then used to sign ephemeral public keys for other schemes: one pair for non-committing encryption, and one pair for ring signatures. This signature binds both ephemeral keys to pk. In an interactive setting, I immediately sends its identity, the ephemeral public keys, and the signature binding them to pk to R as message ψ_1. In a non-interactive setting, I instead uploads this information to a central server as a prekey. Later, when R wishes to send a message to I, the central server relays a prekey to R. I expects only one response for each prekey. Thus, the role of the central server is merely to prevent prekey collisions; the server is not trusted—while it can attack the availability of the protocol by going offline or by distributing non-genuine or repeated prekeys, it is not entrusted with any information that could be used to attack the security of the protocol.

Irrespective of the interactivity mode, R subsequently verifies the consistency of ψ_1 and then uses the ephemeral keys to formulate a response ψ_2 to I. At a high level, this response consists of R's identity, a wrapped session key k, and a signature to authenticate the ciphertext. The session key is first encrypted using dual-receiver encryption for the long-term public keys of I and R. The resulting ciphertext is then encrypted once more using non-committing encryption for npk. Consequently, the session key can only be decrypted by a party knowing $\{nsk, SK_I^{DRE}\}$ or $\{nsk, SK_R^{DRE}\}$. In the normal case (when both I and R are honest), the only party satisfying these requirements is I. In interactive settings, a misinformant in the presence of an online judge will be able to learn k—misinformants simulating I have access to $\{nsk, SK_R^{DRE}\}$, and misinformants simulating R generate k themselves—and thus they can fully simulate conversations.

R also encrypts pk and the identifier for I alongside k. This binds both the non-committing encryption and the dual-receiver encryption to the current session, preventing replay attacks.

As part of the response ψ_2, R also includes a ring signature of the ciphertext. The ring, $\{PK_I^{RS}, PK_R^{RS}, rpk\}$, includes the long-term public keys of both I and R, as well as the ephemeral key rpk associated with pk. In the case where both I and R are honest, R creates the signature using its long-term secret SK_R^{RS}. An honest I is assured that only R could have produced the signature, because I knows that it has not revealed SK_I^{RS} or rsk to any other party. The inclusion of both long-term keys in the ring allows misinformants to simulate signatures for online judges. The inclusion of rpk in the ring allows an offline forger to create transcripts for exchanges between arbitrary parties.

Assuming the existence of a signature scheme, a ring signature scheme that is secure under full-key exposure, a dual-receiver encryption scheme, and a non-committing public-key cryptosystem, Spawn* GUC-realizes $\mathcal{F}_{1psp\text{-}keia}^{\text{IncProc}}$ within the non-erasure $\bar{\mathcal{G}}_{krk}^{\text{Spawn}^*}$-hybrid model with fully adaptive corruptions. For brevity, we omit proof of this claim here; the complete formal proof can be found in the thesis [27, §3.8.4].

6.3 Implications of Incriminating Abort

Like Φ_{dre} and RSDAKE, Spawn* can leak incriminating information if an adversary causes the protocol to abort in a specific manner. When Spawn* is aborted, the adversary can receive a valid ψ_2 message created and signed using R's secret key SK_R^{RS}. There is a critical difference between Spawn*, Φ_{dre}, and RSDAKE in this respect: when incriminating information is released, one of the honest parties (R) does not abort. This leads to a potential attack.

If Mallory, an active network adversary, is attempting to convince Justin that Bob is communicating with Alice using Spawn*, she can cause the protocol to abort. Justin begins by generating ψ_1' using $(npk', nsk') \leftarrow$ NCGen() and sending ψ_1' to Mallory, while ensuring that nsk' is kept private. When Alice sends ψ_1 to Bob, Mallory intercepts this message and replaces it with ψ_1'. Bob responds with ψ_2 containing an encryption γ of session key k and a ring signature σ signed by ring $\{PK_A^{RS}, PK_B^{RS}, rpk\}$. Mallory relays ψ_2 to Justin, but either causes Alice to stall by withholding ψ_2, or causes Alice to abort by delivering ψ_2. Justin now instructs Mallory to corrupt Bob. When Mallory corrupts Bob, she recovers k'. Mallory sends k' and SK_B to Justin. Justin uses his knowledge of nsk and SK_B^{DRE} to decrypt γ, ensuring that $k = k'$. In this case, Justin will be convinced that Bob attempted to communicate with Alice as long as he believes that Mallory did not corrupt Alice or Bob until after ψ_2 was sent. This attack does not apply in settings where it is unreasonable for Justin to make this assumption.

6.4 Weakness of Online Repudiation

When choosing to implement a cryptographic scheme, it is extremely important to consider the security assumptions of the corresponding security proof; when the assumptions do not hold, attacks may be possible. The security of Spawn* relies on the accuracy of the GUC model, which makes an assumption that is not necessarily true in all real-world scenarios. If the distinguishing environment $\mathcal{Z}$ has previously corrupted some party P and gained access to SK_P^{RS}, it may instruct the adversary to cause a corrupted party $R \neq P$ to identify itself as P in response to I. In our security model, we assume that since P is corrupted, $\mathcal{S}$ can also access SK_P^{DRE} in order to decrypt the response sent by R. When this assumption fails in practice, it yields a potential attack in which

$\mathcal{Z}$ can distinguish between simulated and real protocol executions. We will now consider how such an attack might proceed in practice.

Alice is a whistleblower that has previously provided secret information to Bob, a journalist, using Spawn* over the Internet. Justin is an agent for a group that has some leverage over Bob. Justin wishes to prevent further whistleblowing by demanding that Bob asks Alice to send him some new information that will incriminate her. Bob agrees, but refuses to reveal his long-term secret key SK_B^{RS} to Justin since it would allow Justin to impersonate Bob in all conversations. However, Bob does not actually want to incriminate Alice; instead, he would like to secretly simulate Alice.

Bob computes a message ψ_1, and reports to Justin that he has received ψ_1 from Alice. However, Justin has covertly stolen the long-term secrets SK_C from Alice's friend Charlie in the past. Justin constructs a response ψ_2 containing session key k that purportedly comes from Charlie, and signs the ring signature using SK_C^{RS}. He then instructs Bob to send ψ_2 to Alice, along with a message, encrypted under k, asking Alice to meet Charlie for coffee (or some other innocuous message). Justin expects Alice to respond to Charlie's message using a protocol that requires her to know k. If Bob has access to SK_C, as is assumed by the GUC framework, then he can forge the expected response from Alice by recovering k from ψ_2. However, if Bob does not know SK_C, then he cannot simulate a response; Justin has caught Bob attempting to deceive him.

This attack weakens online repudiation in a nuanced manner. If Justin receives a response purportedly from Alice, then he knows that either Bob was communicating with the real Alice, or Charlie's secret key was compromised by Bob (and not only by Justin). If Justin does not receive a response, then he knows that either the real Alice did not respond to the (forged) message from Charlie, or Bob has attempted to deceive Justin by simulating Alice. Note that this situation only occurs when Justin "probes" the honesty of Bob by sending a message from Charlie; Justin does not accomplish his primary objective of incriminating Alice. If Bob predicts that Justin is going to attempt such a probe, he can establish a connection to Alice and relay ψ_2 honestly—but if Bob's prediction is incorrect, then he will unintentionally incriminate Alice. In practice, Justin will always have some uncertainty about the veracity of Alice's responses to Bob.

6.5 Non-interactive Spawn*

The main advantage of Spawn* is that, unlike Φ_{dre} and RSDAKE, it can be used in a non-interactive setting. Non-interactive Spawn* is implemented with the assistance of an untrusted central server tasked with the distribution of prekeys. It is natural to consider this version of the protocol in the context of mobile text messaging. Initially, Alice uses her phone's data connection to connect to a central server operated by the developer of a secure communication app using Spawn*. Alice generates a set of n messages, $\{\psi_{1,1}^{\text{Alice}}, \psi_{1,2}^{\text{Alice}}, ..., \psi_{1,n}^{\text{Alice}}\}$, where each message is a valid message ψ_1 for a Spawn* session. Alice uploads all n message to the central server.[5] Later, Bob uses his phone's data connection to request one of Alice's messages from the server. The server sends an available message $\psi_{1,i}^{\text{Alice}}$, $1 \leq i \leq n$, to Bob and removes it from its list. Bob now uses $\psi_{1,i}^{\text{Alice}}$ to compute a response $\psi_{2,i}^{\text{Alice}}$ to complete the key exchange. He sends $\psi_{2,i}^{\text{Alice}}$ along with his message, encrypted with k, to Alice over his text messaging service. Upon receiving the message, Alice locates the corresponding secret keys for the message to recover k.

[5]The server can deniably authenticate uploaded keys by receiving keys within an interactive Spawn* or RSDAKE session.

Unfortunately, this non-interactive capability comes at a cost: the deniability of the protocol is not as strong as the interactive version. Intuitively, the issue is that ψ_1 is no longer part of a single protocol session; it has been moved into a cross-session global infrastructure. This allows an online judge to request a genuine prekey from the server, overriding any simulated prekey produced by a misinformant. In practice, this means that Spawn* does not provide online repudiation when R attempts to simulate I in the non-interactive setting. However, I still maintains online repudiation in this setting—I can reliably simulate a response from any party, even in the presence of online judges—and all other security properties of the protocol continue to hold. Thus, non-interactive Spawn* still provides stronger deniability guarantees than triple Diffie-Hellman (3-DH), the current (non-interactive) TextSecure key exchange protocol.

6.6 Conjecture: TextSecure Iron Triangle

Given the incomplete online repudiation of Spawn* when it is used non-interactively, an obvious question to ask is whether the protocol can be modified to address these problems. We may also wonder more generally about all key exchanges suitable for use in the TextSecure setting. We define a *TextSecure-like key exchange* as a one-round key exchange protocol in which the initiator I does not initially know the identity of the responder R. Our results lead us to an unfortunate suspicion about the nature of such protocols: any TextSecure-like key exchange cannot simultaneously provide non-interactivity, (weak) forward secrecy, and online repudiation with respect to R simulating I.

Intuitively, this conflict arises from the set of secrets required to recover the session key k from the protocol transcript. In general, both I and R may have short-term secrets (sk_I and sk_R, respectively) and long-term secrets (SK_I and SK_R, respectively). In a non-interactive setting, R cannot simulate I's generation of sk_I to an online judge (for the reasons given in Section 6.5), and the online judge can insist on generating sk_R itself. Consequently, the only secret information known only by R in this case is SK_R. If R is able to recover k from the transcript, then this implies that the protocol does not have forward secrecy (because only long-term secrets are required to recover k). If R is not able to recover k from the transcript, then this implies that the protocol lacks online repudiation (because R cannot simulate I's subsequent use of k). Additionally, there is no way to force the judge to reveal any secrets to R since the judge can always insist on the use of a secure multiparty computation protocol to generate any required response.

6.7 A Practical Relaxation: Spawn

The security properties of Spawn* hold in a very strong threat model: the adversary can adaptively corrupt parties, and no information can ever be erased. In practice, these assumptions may not hold. If either assumption is removed, then the security model will admit a modified version of Spawn* with substantially increased performance. Spawn is a protocol that is equivalent to Spawn*, except that it replaces the use of non-committing encryption in ψ_2 with a standard public-key cryptosystem. PKGen is used to generate an ephemeral key pair (epk, esk) rather than the (npk, nsk) pair produced by NCGen in Spawn*. The motivation for this modification is that non-committing encryption schemes are extremely expensive compared to standard public-key cryptosystems (e.g., the NCE scheme described by Walfish [29] makes $2\lambda n$ calls to the PKEnc function of an underlying 1-bit PKE scheme to encrypt an n-bit message with security parameter λ).

In practice, it is reasonable to accept this weaker threat model in many common environments. Security and privacy tools such

as hard drive encryption utilities and secure messaging tools commonly assume that cryptographic keys can be limited to RAM storage, and RAM can be securely erased while a machine is uncorrupted. It is also reasonable to assume that corruptions are not fully adaptive. When Spawn is used interactively, I can easily erase esk if a timely response is not received from R (e.g., if an adversary prevents delivery of ψ_2 in an attempt to cause I to retain esk for later corruption). In the non-interactive setting, online repudiation of Spawn* is already weakened (see Section 6.5); there are no practical situations in non-interactive Spawn* that actually require the use of NCEqv for simulation. For this reason, the use of Spawn instead of Spawn* causes no loss of deniability beyond that already incurred due to use in a non-interactive environment.

6.8 Bootstrapping Axolotl

One of the most innovative features of TextSecure is its per-message forward and backward secrecy [28]; if keys are compromised during a conversation, messages sent before *or* after the compromise cannot be decrypted by the adversary. TextSecure achieves these properties by using the 3-DH DAKE to initialize Axolotl, a scheme that refreshes cryptographic keys over time [21]. While 3-DH provides deniability against offline judges, it does not defend against online judges. Spawn can improve the deniability properties of TextSecure by replacing the 3-DH key exchange.

When Axolotl begins, I and R perform a 3-DH key exchange to obtain a shared secret k. A key derivation function is used to derive a variety of keys from k that are used internally by Axolotl. Spawn (or Spawn*) can be used instead of 3-DH to share k, which can then be passed through the key derivation function in the same manner. One complication that arises comes from the fact that 3-DH is a contributory DAKE, while Spawn is non-contributory. Axolotl normally encrypts the first message in the conversation using, in part, an ephemeral secret key generated by R, for which there is no analog in Spawn. Luckily, this is easily solved by encrypting the first message using a key derived solely from k; the details of Axolotl ensure that subsequent messages are encrypted using freshly generated ephemeral keys. The thesis describes the complete technical details of incorporating Spawn into TextSecure [27, §3.8.10].

7. IMPLEMENTATION

Currently, there exists a disconnect between secure messaging system developers and the academic community; there is an abundance of solutions described in the literature that are never implemented [28]. It has become clear that describing a new system and writing security proofs, while necessary, are insufficient for making cryptography usable; we need to do more if we want actual users to benefit from our schemes. One way to bridge this gap is to provide open implementations of our designs to encourage use by developers of consumer security products. We developed open-source implementations[6] of the key exchanges presented in this work using the Go programming language [14].

Our primary development objective was to implement Φ_{dre} with non-interactive DRE, Φ_{dre} with interactive DRE (hereafter referred to as Φ_{idre}), RSDAKE, and Spawn. Since the specialized cryptosystems used by these protocols lack widely available implementations, we also developed implementations of these underlying schemes as part of our development effort. In summary, we produced the following libraries:

- Pairing-Based Cryptography Library [17] wrapper for Go;
- Shacham-Waters [26] ring signatures;
- HORS [23] one-time signatures with HORS+ [30] improvement;
- Cramer-Shoup cryptosystem [10], both in finite fields of prime order and elliptic curve groups;
- Chow-Franklin-Zhang [9] BDDH-based DRE;
- Interactive DRE (defined in Section 4.2);
- Φ_{dre} [29] (the first public implementation);
- RSDAKE (defined in Section 5.2);
- Spawn (defined in Section 6.7).

We used the Chow-Franklin-Zhang non-interactive DRE scheme in Φ_{dre} and Spawn, the Shacham-Waters ring signature scheme in RSDAKE and Spawn, the Cramer-Shoup cryptosystem in Φ_{dre} and Spawn, and ECDSA [18] in RSDAKE and Spawn. All of the implemented schemes are provably secure in the standard model (i.e., they do not require random oracles). Our implementations each come with predefined parameters to approximate security levels between 80 and 256 bits. Precise details about the operation of our libraries, and the exact standard-model security assumptions made by each, are available in the thesis [27, §4].

8. EVALUATION

To compare the performance of the key exchange implementations, we instantiated a simulation of an interactive session between two parties over the Internet. This simulation modeled a duplex connection with configurable transmission latency and bandwidth restrictions. We evaluated the performance of four protocols: Φ_{dre}, Φ_{idre}, RSDAKE, and Spawn. We tested each protocol in a variety of simulated network conditions at 112-, 128-, and 192-bit approximate security levels. We simulated message latencies at 0, 50, 100, 300, 1000, 2000, 5000, and 10000 milliseconds. We simulated communication channel bandwidth at 10 Gib/s, 100 Mib/s, 20 Mib/s, 5 Mib/s, 500 Kib/s, and 50 Kib/s. We performed each test 200 times on 3.6 GHz processor cores with access to RAM providing 15 GiB/s read and write speeds with 63 ns latency. All graphs in this section make use of logarithmic vertical axes and error bars. The error bars, which denote the standard error of the mean, are typically too small to see.

8.1 Space Complexity

All four schemes transmit different amounts of data during the protocol session. The amount of data transmitted depends only on the choice of protocol and the security level; it does not depend on the speed of the network connection. Figure 4 shows the total amount of data transmitted by each protocol during a session; this total represents the sum of the number of bytes written at the application layer by each party. All schemes are relatively expensive compared to a simple SIGMA protocol; all four schemes require at least 4 KiB to complete a session with at least 112 bits of security. However, both Φ_{dre} and Spawn require significantly more data transmission than Φ_{idre} or RSDAKE; additionally, Φ_{dre} transmits approximately three times more data than Spawn. The reason for this disparity is the use of the HORS+ one-time signature scheme by the non-interactive DRE implementation. Since Φ_{dre} makes use of three DRE encryptions and Spawn makes use of only one, Φ_{dre} requires nearly three times more data than Spawn to complete the exchange. The use of interactive DRE in Φ_{idre} dramatically reduces the data costs of the protocol; Φ_{idre} consistently uses the least data of all four protocols.

While RSDAKE uses nearly as little data as Φ_{idre} for the 112-bit security level, its costs increase much faster as the security level is

[6]The resulting libraries can be found at `https://crysp.uwaterloo.ca/software/`.

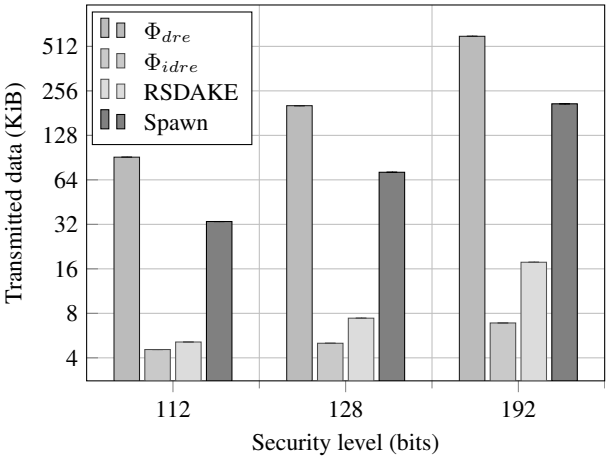

Figure 4: The amount of data transmitted increases significantly with higher security levels. Φ_{dre} and Spawn require significantly more transmissions than Φ_{idre} or RSDAKE.

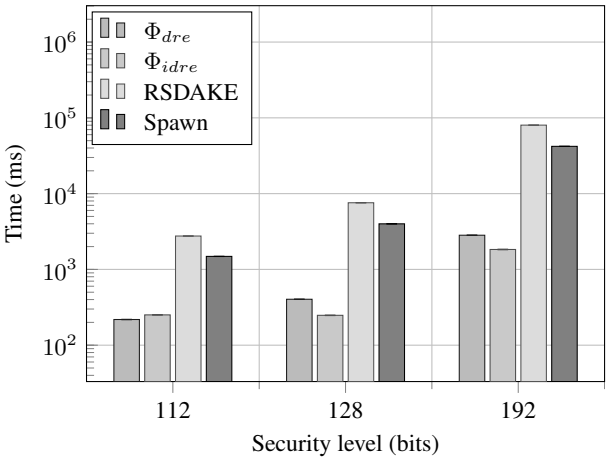

Figure 5: Over a high-bandwidth connection with no latency, the cryptographic overhead of each protocol is clear. The use of ring signatures negatively affects RSDAKE and Spawn.

increased. Since the Shacham-Waters ring signature scheme used by RSDAKE requires composite-order bilinear groups, the relative ease of the integer factorization problem requires that the size of group elements increases with approximately the cube of the security level. As a result, the two ring signatures exchanged within RSDAKE rapidly grow in size with heightened security. Nonetheless, RSDAKE never approaches the transmission costs of Φ_{dre} or Spawn, even at the 192-bit security level.

8.2 Time Complexity vs. Security Level

As the desired security level increases, all four schemes require increasingly expensive cryptographic operations. To understand the impact of security levels on the time complexity of the algorithms, we focus on the simulation with minimal impact from network conditions. Figure 5 shows the amount of time required to complete a session of each protocol when the parties are connected through a 10 Gib/s channel with no latency; the resulting delays are directly indicative each scheme's cryptographic overhead.

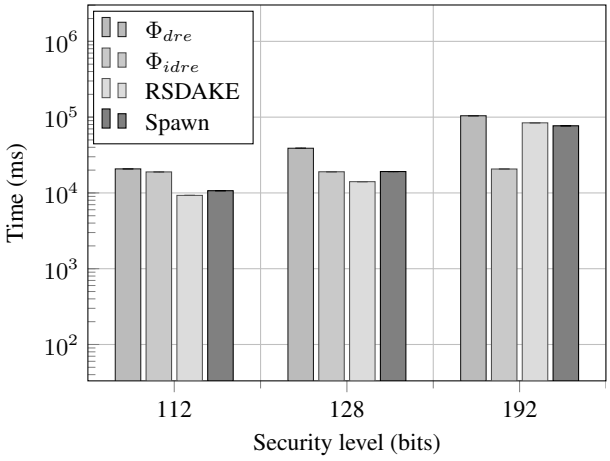

Figure 6: Over a low-bandwidth and high-latency connection, the network significantly affects performance. RSDAKE and Spawn perform the best at 112- and 128-bit security levels.

As shown in Figure 5, RSDAKE and Spawn are roughly an order of magnitude more expensive than Φ_{dre} and Φ_{idre}. The ring signature scheme used by both RSDAKE and Spawn is to blame for this disparity. As we mentioned in Section 8.1, the Shacham-Waters scheme used in our implementation makes use of composite-order bilinear groups. Operations in this group setting are much more expensive than operations in the prime-order elliptic curve groups used by Φ_{dre} and Φ_{idre}. RSDAKE suffers more from this expense than Spawn because RSDAKE requires two ring signatures.

Both Φ_{dre} and Φ_{idre} are extremely computationally efficient, requiring less than one second to complete at the 112- and 128-bit security levels. However, in this fast network environment, the interactive DRE scheme used by Φ_{idre} scales better than the Chow-Franklin-Zhang scheme used by Φ_{dre}. At the 128- and 192-bit security levels, Φ_{idre} requires the least amount of time to complete. This performance improvement can be attributed to the direct use of elliptic curve groups by the Cramer-Shoup scheme in Φ_{idre}, rather than the use of pairing-based cryptography in Φ_{dre}.

It is also useful to understand how the schemes react to poor network conditions. Figure 6 shows the total time required to complete a session of each protocol when the parties are communicating across a 50 Kib/s connection with 2 seconds of latency. This simulation models an extremely poor network environment that is effectively a worst-case scenario for the protocols; the primary use of this model is to provide insight into how the protocols behave under difficult network conditions.

The poor performance of the network connection depicted in Figure 6 dominates the cost of all four protocols. Despite requiring nine message flows to complete, Φ_{idre} performs comparatively well in this high-latency environment. Φ_{idre} is the most scalable protocol; since it only requires operations in small elliptic curve groups, the cryptographic overhead is relatively constant. Φ_{dre} performs the worst at all security levels since it makes use of the Chow-Franklin-Zhang DRE scheme three times, which imposes high bandwidth costs. Spawn generally performs well since it only requires two message flows to complete, but it is still slower than RSDAKE at 128- and 192-bit security levels due to its use of the Chow-Franklin-Zhang DRE scheme. RSDAKE is the most efficient protocol at the 112- and 128-bit security levels. At the 192-bit security level, the performance of RSDAKE and Spawn is impacted by the computational costs of the Shacham-Waters ring signature

scheme. Consequently, Φ_{idre} performs significantly better than all other protocols at the 192-bit security level.

In the thesis, we examine the scalability of the algorithms with respect to network latency and bandwidth at various security levels [27, §4.3]. As a two-flow protocol, Spawn scales the best as latency increases. Φ_{idre} scales the worst with increasing latency since it requires nine flows. The performance of Φ_{dre} and Spawn begin to deteriorate rapidly as channel bandwidth decreases below 5 Mib/s, especially at higher security levels.

9. CONCLUSION

When choosing a protocol to use in a real-world application, developers should consider their security and performance needs. Φ_{dre}, RSDAKE, and Spawn have different security and usability properties, and thus are best suited for different environments.

We have introduced Spawn, the first protocol with strong deniability properties and forward secrecy that can be used in non-interactive environments. Spawn also requires the fewest number of flows. Although non-interactive Spawn does not provide online repudiation with respect to R simulating I (see Section 6.5), it still provides improved deniability properties compared to 3-DH. When a practitioner only needs to support interactive environments, more choices are available: Spawn is useful in interactive environments where the weaknesses described in Section 6.3 and Section 6.4 are not a concern; Φ_{dre} and our newly defined RSDAKE scheme can both be used interactively and offer the same security properties, but RSDAKE offers some additional features. Unlike Φ_{dre}, RSDAKE is a contributory key exchange proven secure in the post-specified peer setting.

It is also important to understand how the schemes perform in practice under various network conditions. Our evaluation presented in Section 8 is meant to serve as a guideline for real-world performance expectations. When implemented using primitives secure under only standard-model assumptions, Φ_{idre} and RSDAKE are the best choices for bandwidth-constrained environments such as mobile data connections. Φ_{dre} and Φ_{idre} are better suited for use over large and fast connections. Only Spawn supports non-interactive environments, irrespective of their network characteristics. If a practitioner is willing to make use of schemes that depend on random oracles for security, then the performance of all four protocols can be greatly improved.

Deniability of secure messaging schemes remains a research area with many unsolved problems. The most appropriate definition of deniability to use when constructing protocols is not yet agreed upon; specifically, very few publications consider online repudiation during analysis of their designs. While we suspect that weak forward secrecy and online repudiation are mutually exclusive in the non-interactive setting, the conjecture in Section 6.6 remains unproven. Finally, although we provide proof-of-concept implementations of our new DAKE protocols, adoption by end-user tools may be encouraged by integrating these implementations with a higher-level popular cryptographic library.

10. ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their insightful comments and feedback. We gratefully acknowledge the support of NSERC and the Ontario Research Fund.

11. REFERENCES

[1] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. In *Advances in Cryptology–EUROCRYPT*, pages 139–155. Springer, 2000.

[2] A. Bender, J. Katz, and R. Morselli. Ring Signatures: Stronger Definitions, and Constructions without Random Oracles. In *Theory of Cryptography*, pages 60–79. Springer, 2006.

[3] N. Borisov, I. Goldberg, and E. Brewer. Off-the-Record Communication, or, Why Not To Use PGP. In *Workshop on Privacy in the Electronic Society*, pages 77–84. ACM, 2004.

[4] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Foundations of Computer Science*, pages 136–145. IEEE, 2001.

[5] R. Canetti. Universally Composable Signature, Certification, and Authentication. In *Computer Security Foundations Workshop*, pages 219–233. IEEE, 2004.

[6] R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively Secure Multi-party Computation. Technical report, Massachusetts Institute of Technology, 1996. `http://theory.csail.mit.edu/ftp-data/pub/people/oded/dynamic.ps`.

[7] R. Canetti and H. Krawczyk. Security Analysis of IKE's Signature-Based Key-Exchange Protocol. In *Advances in Cryptology–CRYPTO 2002*, pages 143–161. Springer, 2002.

[8] S. G. Choi, D. Dachman-Soled, T. Malkin, and H. Wee. Improved Non-Committing Encryption with Applications to Adaptively Secure Protocols. In *Advances in Cryptology–ASIACRYPT 2009*, pages 287–302. Springer, 2009.

[9] S. S. Chow, M. Franklin, and H. Zhang. Practical Dual-Receiver Encryption. In *Topics in Cryptology–CT-RSA 2014*, pages 85–105. Springer, 2014.

[10] R. Cramer and V. Shoup. A Practical Public Key Cryptosystem Provably Secure against Adaptive Chosen Ciphertext Attack. In *Advances in Cryptology–CRYPTO'98*, pages 13–25. Springer, 1998.

[11] I. Damgård and J. B. Nielsen. Improved Non-Committing Encryption Schemes Based on a General Complexity Assumption. In *Advances in Cryptology–CRYPTO 2000*, pages 432–450. Springer, 2000.

[12] Y. Dodis, J. Katz, A. Smith, and S. Walfish. Composability and On-Line Deniability of Authentication. In *Theory of Cryptography*, pages 146–162. Springer, 2009.

[13] A. Fiat and A. Shamir. How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology–CRYPTO'86*, pages 186–194. Springer, 1987.

[14] Go Project. The Go Programming Language. `https://golang.org/`, 2009. Accessed 2015-04-13.

[15] M. Hearn. Value of deniability. Mailing list discussion, `https://moderncrypto.org/mail-archive/messaging/2014/001173.html`, 2014. Accessed 2015-04-02.

[16] H. Krawczyk. SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and its Use in the IKE protocols. In *Advances in Cryptology–CRYPTO 2003*, pages 400–425. Springer, 2003.

[17] B. Lynn. The Pairing-Based Cryptography Library. `https://crypto.stanford.edu/pbc/`, 2006. Accessed 2015-04-13.

[18] U. D. of Commerce / National Institute of Standards & Technology. Digital Signature Standard (DSS), 2013.

[19] Open Whisper Systems. Open WhisperSystems. https://www.whispersystems.org/, 2013. Accessed 2014-11-02.

[20] Open Whisper Systems. Open Whisper Systems partners with WhatsApp to provide end-to-end encryption. https://www.whispersystems.org/blog/whatsapp/, 2014. Accessed 2014-12-23.

[21] T. Perrin. Axolotl Ratchet. https://github.com/trevp/axolotl/wiki, 2013. Accessed 2014-11-02.

[22] C. Rackoff and D. R. Simon. Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack. In *Advances in Cryptology–CRYPTO'91*, pages 433–444. Springer, 1992.

[23] L. Reyzin and N. Reyzin. Better than BiBa: Short One-time Signatures with Fast Signing and Verifying. In *Information Security and Privacy*, pages 144–153. Springer, 2002.

[24] R. L. Rivest, A. Shamir, and Y. Tauman. How to Leak a Secret. In *Advances in Cryptology–ASIACRYPT 2001*, pages 552–565. Springer, 2001.

[25] C.-P. Schnorr. Efficient Signature Generation by Smart Cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[26] H. Shacham and B. Waters. Efficient Ring Signatures without Random Oracles. In *Public Key Cryptography*, pages 166–180. Springer, 2007.

[27] N. Unger. Deniable Key Exchanges for Secure Messaging. Master's thesis, University of Waterloo, 2015. http://hdl.handle.net/10012/9406.

[28] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith. SoK: Secure Messaging. In *Symposium on Security and Privacy*. IEEE, 2015.

[29] S. Walfish. *Enhanced Security Models for Network Protocols*. PhD thesis, New York University, 2008.

[30] J. Zhang, J. Ma, and S. Moon. Universally composable one-time signature and broadcast authentication. *Science China Information Sciences*, 53(3):567–580, 2010.

[31] H. Zhu, T. Araragi, T. Nishide, and K. Sakurai. Universally Composable Non-committing Encryptions in the Presence of Adaptive Adversaries. In *e-Business and Telecommunications*, pages 274–288. Springer, 2012.