An Approximate Tamarin Analysis of the X3DH Key Agreement Protocol

Ronak Malik
Harvard College
Cambridge, Massachusetts, USA
ronakmalik@college.harvard.edu

Albert Qi
Harvard College
Cambridge, Massachusetts, USA
albertqi@college.harvard.edu

Justin Ye
Harvard College
Cambridge, Massachusetts, USA
justinye@college.harvard.edu

ABSTRACT

As technology becomes increasingly integral in our daily lives, more and more people rely on messaging apps to communicate. However, it is also paramount that communication through these apps is secure and privacy is maintained; users need to be certain that their messages to others will not be intercepted. Signal is one such platform that claims it has strong security guarantees, including both forward secrecy and cryptographic deniability. In order to achieve these guarantees, it utilizes the X3DH key agreement protocol.

How can we be absolutely certain that the X3DH protocol does in fact achieve its guarantees, though? To test this, we utilize Tamarin, a security protocol verification tool, in order to formally verify the properties of an approximate version of the X3DH protocol. Through our implementation, we verify that both forward secrecy and cryptographic deniability are achieved.

All source code can be accessed at https://github.com/albertqi/x3dh-tamarin.

1 INTRODUCTION

People have become increasingly reliant on messaging apps in their day-to-day lives. The Signal app is one such messaging service, and it makes strong promises about the security of its platform. One particularly important promise is the claim that nobody besides the intended recipient, including Signal itself, is able to read messages on the end-to-end encrypted platform. These promises stem from Signal's use of the X3DH, or Extended Triple Diffie-Hellman, key agreement protocol.

X3DH is a key agreement protocol that establishes a shared secret key via public and private key pairs. There are three parties: Alice, Bob, and a server. The keys that are used are IK_A, EK_A, IK_B, SPK_B, and OPK_B, which represent Alice's identity key, Alice's ephemeral key, Bob's identity key, Bob's signed prekey, and Bob's one-time prekey, respectively. The identity keys are long-term, Bob's signed prekey is changed periodically, and both Alice's ephemeral key and Bob's one-time prekey are changed during each protocol run. Note that each key has a public and private variant.

The protocol operates in three phases. First, Bob sends his identity key and prekeys to the server. Second, Alice fetches these keys and combines them with her keys to generate a shared secret key. She then sends over her keys and an initial message to Bob. Finally, Bob fetches Alice's keys and combines them with his keys to generate the same shared secret key. He can use this to decrypt the initial message sent by Alice.

Signal claims that X3DH guarantees both forward secrecy and cryptographic deniability. Forward secrecy is a property in which the compromise of long-term keys does not compromise past session keys. Thus, even if an adversary were to get ahold of Alice's and Bob's current private keys, then they would still not be able to decrypt past communications. This is achieved through the constant regeneration of certain public-private key pairs. Cryptographic deniability is a property in which Alice and Bob can deny that they ever communicated with each other, even if the private keys are compromised. After the communication has taken place, Alice and Bob cannot prove that the communication ever occurred.

To explore the X3DH protocol and to reason about its properties, we implement the X3DH protocol in Tamarin, a security protocol verification tool. Implementing the protocol allows us to understand some of the lower level details of the protocol as well as examine their security promises under various conditions and models. Such work helps us understand how well X3DH and Signal's security promises hold up formally.

Some of the main challenges in implementing the X3DH protocol in Tamarin include (1) the lack of elliptic key generation and HKDF hashing capabilities, (2) reasoning about the logical differences between the true X3DH protocol and our implementation of it, and (3) the slow Tamarin prover speed.

(1) The X3DH protocol uses elliptic key generation and HKDF hashing, but Tamarin does not support either of these. To handle this issue, we instead utilize Tamarin's modular exponentiation and hash functions, under reasoning that the security guarantees should still hold.

(2) It is challenging to reason about exactly what changes between the true X3DH protocol and our implementation of the protocol.
(3) The Tamarin prover speed is rather slow, especially given the complexity of the protocol. This makes testing fairly challenging.

Nonetheless, we still learn a lot from our implementation of X3DH. Namely, based on our model of the protocol, it does guarantee forward secrecy and cryptographic deniability. Even when private keys are leaked, past messages are still not decryptable by an adversary. Furthermore, Alice and Bob can plausibly deny that they ever communicated with each other, even if their keys are leaked.

2 BACKGROUND

We begin by describing the general idea of Diffie-Hellman secret-sharing. This is a method for two parties who can only communicate over a public channel but want to establish a shared key with which to encrypt their data to ensure that their messages are private. Traditionally, parties Alice and Bob agree on a generator g and a prime p over the public channel, and each also has their own secret key sk_a and sk_b, which they do not share. Each user then computes $x = g^{sk_a} \bmod p$ and $y = g^{sk_b} \bmod p$, respectively. These represent Alice and Bob's public keys. Alice can send x to Bob, and vice versa, and then Alice and Bob respectively compute $k_a = y^{sk_a} \bmod p$ and $k_b = x^{sk_b} \bmod p$. It can be shown that $k_a = k_b$, and so both parties now agree on a shared secret key. Furthermore, the inversion of the exponentiation which is performed to compute these keys is computationally difficult; thus, other parties are unable to arrive at the same secret key without one of Alice's or Bob's private keys [2].

Now, we will describe the X3DH protocol as defined by the Signal app. There are three key parties involved in the protocol. Alice and Bob are two users who want to establish a shared secret key over public communication channels for encryption; however, Bob may not be online during this key-agreement protocol. The server acts as an intermediary between the two and allows Bob to both publish persistent messages and retrieve messages from Alice at a later time [6].

During the X3DH protocol, both Alice and Bob generate several public-private key pairs using elliptic-curve cryptography, which is a set of algorithms that can be efficiently and securely applied to Diffie-Hellman protocols [4]. Specifically, Alice and Bob both generate long-term identity keys IK_A and IK_B. Bob generates a set of one-time keys (OPK_B), one of which is used for every protocol run; likewise, Alice generates an ephemeral public-private key pair EK_A for each protocol run as well. Last, Bob has a signed prekey SPK_B which last for longer than the one-time key but changes periodically (on the order of weeks to months), unlike the identity key.

The X3DH protocol is split into three phases. First, Bob publishes a bundle of his public keys to the server, including his public identity key, signed prekey, and a set of one-time prekeys, indexed as $OPK_B^1, OPK_B^2, \ldots$ and so on. Additionally, Bob publishes a prekey signature which signs SPK_B with Bob's private identity key in order to verify his identity among this key bundle package. The signature uses Signal's XEdDSA protocol [7].

Then, when Alice wants to initiate communication with Bob, she initializes a new ephemeral key pair EK_A and fetches the bundle of keys from the server. She can then use the Diffie-Hellman elliptic curve function to compute several outputs: $s_1 = DH(IK_A, SPK_A)$, $s_2 = DH(EK_A, IK_B)$, $s_3 = DH(EK_A, SPK_B)$, and $s_4 = DH(EK_A, OPK_B)$, where DH is a Diffie-Hellman encryption function that generates a shared secret between Alice's private keys and Bob's public keys. Then, she concatenates s_1, s_2, s_3, s_4 and applies the HKDF algorithm to it; this is essentially a hashing function which creates a shared secret and also allows for future shared secrets to be built on its output [3]. The output of this HKDF function is notated as SK, the shared secret key for encryption. Using this, Alice can encrypt both her and Bob's public identity keys. Next, Alice deletes her ephemeral private key and s_1, s_2, s_3, s_4. She can then send Bob a message containing her public identity and ephemeral keys, a list of which of Bob's keys were used (i.e., Bob's one-time key), and ciphertext encoded using SK, where the encodings of her and Bob's identity keys is some associated data. Alice's message can be sent to the server, which stores the message until Bob is online.

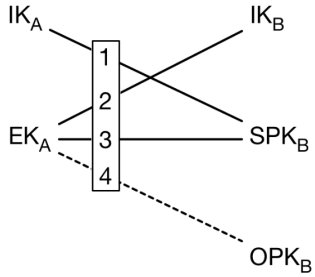

Figure 1: Diagram Showing Pairings of Alice's and Bob's Keys Contributing to the Secret Key in X3DH [6]

Finally, when Bob receives the message from the server, he can take Alice's public identity and ephemeral keys and create the same s_1, s_2, s_3, s_4 using his private identity key, signed prekey, and the one-time key as described by Alice. Bob is able to use these values to recompute SK in the same way that Alice did, and he can then compute the associated data

using Alice's and Bob's public identity keys and *SK*. Then, Bob is able to use the associated data and shared secret key to decipher the initial message and can respond by encrypting with *SK*. If this decryption fails, then Bob aborts the protocol; if it succeeds, then Bob removes the used-up one-time key from the server (note that these are also replenished if all are used up) [6].

Signal claims this protocol has strong security guarantees. Publicly, Signal markets the fact that nobody, including itself (i.e., the server), can read any messages between two parties like Alice and Bob [8]. In their description of X3DH, they state that neither Alice nor Bob has "a publishable cryptographic proof of the contents of their communication or the fact that they communicated", suggesting that if one party is compromised, then the attacker that compromised them cannot prove the existence of communications with another party. Furthermore, X3DH demonstrates forward secrecy, where if one-time private keys or Bob's identity private key are compromised, then past messages cannot be decrypted. If private prekeys are compromised while their public values are still stored and available for use on the server, however, then future messages may be vulnerable until these keys are replaced. The protocol is also resilient against a malicious server: under the assumption that Alice and Bob are able to authenticate each other's public keys offline or through some non-server channel and under the assumption that the server does allow messages to go between Alice and Bob at all, then the server can at most weaken, but not fully compromise, X3DH's forward secrecy properties [6].

3 DESIGN

We describe the following assumptions that we work with for our implementation of the X3DH protocol.

3.1 Model Assumptions

Our implementation of X3DH in Tamarin works off a few model assumptions which are key to the protocol:

(1) Parties such as Alice and Bob are able to confirm each others' public identity keys through some authenticated channel or through some manual mechanism. This confirms the identity of the other party when messaging over other channels and avoids adversaries lying about said values, so it provides greater guarantees about to whom each party is talking [6].
(2) While the server may be an adversary to both Alice and Bob, it maintains the integrity of communication between the two (i.e., it does not tamper with messages other than when specified by the threat model). This also means that it does not prevent messages from going between Alice and Bob.
(3) Inverting the Diffie-Hellman equation (both the standard modular exponentiation problem and the elliptic curve function used in X3DH) is a sufficiently difficult problem for encryption.

Unfortunately, Tamarin does not provide support for all of the functions which are used in the X3DH protocol. In particular, we found that Tamarin lacks support for both elliptic curve cryptography and for the specific key derivation function (HKDF) which is used to generate the shared secret key. With this in mind, we design a modified version of the X3DH protocol which we believe encapsulates the spirit of the protocol with the tools provided by Tamarin. It differs from the protocol described above in the following ways:

(1) Rather than generating key pairs using the elliptic curve function and applying elliptic curve multiplication as the function to generate shared secrets between parties' private and public keys, we use the built-in modular exponentiation tools to perform Diffie-Hellman. Each key pair has a unique constant as its generator.
(2) We use the built-in hash function in place of the HKDF function. The main purpose of the HKDF function is to (a) hash secret outputs into a shared secret key for encryption, and (b) allow for the creation of subsequent keys based on the initial one. While point (b) is particularly important for Signal, which wants to continually and rapidly maintain security for future messages and sessions, we are mostly interested in the X3DH key-exchange protocol in isolation which is only used for the initial message and handshake. Furthermore, X3DH's protocol also allows for the continued use of a single, initial shared secret key. Thus, for the purposes of our Tamarin implementation, we use the built-in hash function with shared Diffie-Hellman outputs and use it as a single, time-stationary encryption key, rather than using HKDF.
(3) We have implementations which do and do not include the optional one-time key.
(4) The protocol in Tamarin is not specifically designed for an asynchronous setting. It does not prescribe a time period over which the protocol occurs, and so it is agnostic to whether users are synchronized or not.

3.2 Threat Model

Since X3DH is used to securely establish a shared secret, the primary threat model will be defined around ways in which an adversary can achieve unauthorized access to messages sent between the agents involved in the handshake. We assume that the agents in the handshake are remote, so they must communicate over an insecure network. This means that any messages sent over the network are available to

the adversary. We model this with `In()` and `Out()` facts in Tamarin. We assume that any information passed to `Out()` is added to the adversary's knowledge base.

Since X3DH is designed to work when one party is offline, we assume that all messages passed between agents are relayed by a server, and information such as identity keys, prekeys, and signatures are stored and delivered when the receiving agent comes back online. However, we choose not to actually model the server in Tamarin and instead treat it as a standard adversary that is capable of snooping on unsecured network traffic. We do this because a byzantine or adversarial server that randomly tampers with messages would just cause the handshake to terminate rather than causing any information leaks. Therefore, for simplicity, we assume that the server does its job as a relay but may still snoop on messages with the intent of stealing information.

With regards to additional capabilities, we do not assume that the server is able to observe additional information about any participants, such as about computation time or memory resource usage. We also assume that the adversary cannot persuade either party to act against their own interests or in a specific way, nor can the adversary be present at the confirmation of public identities which Alice and Bob perform offline. In other words, we assume that an adversary has complete knowledge of the contents passed through the remote communication channel but no additional information about either party.

4 EVALUATION

4.1 Shared Key and Message Secrecy

Our first evaluation metric is ensuring that our implementation of the X3DH handshake maintains message secrecy throughout the protocol. This means that at no point should the adversary be able to deduce the shared secret or any messages shared between the two agents.

To prove this, we release the shared message as a fact and then check that it is never available to the adversary with the following lemmas, which verify that the shared keys and the message are never leaked.

```
lemma ResponderKeySecrecy:
    "All k #t. ResponderKey(k) @ #t
    ==> not Ex #x. K(k) @ #x"

lemma InitiatorKeySecrecy:
    "All k #t. InitiatorKey(k) @ #t
    ==> not Ex #x. K(k) @ #x"

lemma SendMsgSecrecy:
    "All m #t. MessageWasSent(m) @ #t
    ==> not Ex #x. K(m) @ #x"
```

As expected, under a simpler model with no one-time prekeys, the messages and keys remain secret and the lemmas are satisfied. When one-time keys are included, however, we are unable to evaluate the key secrecy lemmas due to Tamarin constraints and limitations on personal computation power; the message secrecy lemma still holds. However, we expect the key secrecy lemmas to still hold given that they hold for both the standard Diffie-Hellman procedure and for the simpler version of X3DH. The addition of one-time keys should only add security as another layer of encryption.

4.2 Forward Secrecy

X3DH additionally provides guarantees about the secrecy of messages given that a long term key is leaked. In our case of just examining the handshake, this means that the initial message cannot be deduced even if the private keys of the two agents are leaked after the handshake is complete. This is achieved through the addition of the ephemeral and one time keys that are added to the exchange. We assume that these keys do not leak and are deleted after use.

We formally prove our model maintains forward secrecy by adding a rule which leaks the long-term identity keys of both Alice and Bob and then check to see if the adversary can deduce the message after the leak. The rule to leak the keys looks as follows:

```
rule LeakPrivateKeys:
    [ !PrivateIdentityKey($A, ~pIKA),
      !PrivateIdentityKey($B, ~pIKB),
      !PrivateSignedPrekey($B, ~pSPKB) ]
    --[ Leak() ]->
    [ Out(<~pIKA, ~pIKB, ~pSPKB>) ]
```

Next, we also create a lemma which evaluates forward secrecy. If a leak occurs after a message is sent, then that message cannot be decrypted:

```
lemma ForwardSecrecy:
   "All m #i #j. #i < #j & MessageWasSent(m) @
    #i & Leak() @ #j ==> not Ex #x. K(m) @ #x"
```

This lemma holds, which is expected from the X3DH protocol. Indeed, the introduction of ephemeral and one-time keys helps us achieve forward secrecy.

4.3 Mutual Authentication and Cryptographic Deniability

Included in the X3DH handshake is the ability for each party to mutually authenticate the other party, which is usually done through a different mechanism such as RSA in standard Diffie-Hellman. Here, authentication is done by including a prekey signature that is sent by Bob and verified by Alice.

Rather than writing a lemma to prove that the agents are able to authenticate each other, we found that using restrictions on traces provides better performance while maintaining the semantics of the property. To prove that authentication holds during the X3DH protocol, we restrict the space of possible traces to the ones that successfully verify the prekey signature with the public identity key received in the initial bundle or second stage of the handshake. This is done through the Tamarin `restriction` rule as shown below:

```
restriction Eq:
   "All t1 t2 #x. Eq(t1, t2) @ #x ==> t1 = t2"
```

Then, we enforce the authentication by using the Tamarin built-in `verify` function to check that the signature matches the identity public key received from the other agent. Finally, we introduce an `Executability` lemma that ensures that a trace exists which successfully completes the handshake. This ensures that it is possible for both agents to authenticate each other and for the handshake to complete successfully.

It is important to note that authentication here requires some out of band communication (e.g., meeting up in person) to verify the identity keys of each agent. Otherwise, it is possible for the server to impersonate other agents by providing a false identity key and prekey signature. For the sake of this model, we assume that the server does not do this or that the agents independently verify each others' identity.

Mutual authentication using Diffie-Hellman theoretically also provides cryptographic deniability, meaning that a third-party cannot deduce who sent a message if they did not observe the agent sending the message. This is achieved by performing the authentication during the handshake phase rather than explicitly signing each message. If each message was signed with the identity key of an agent, then that proves that an agent sent that message. Instead, the shared secret used to encrypt each message is built from a Diffie-Hellman parameter that can only be acquired when the two agents mutually authenticate during the handshake (this is the signed prekey step described in Section 2). Since each message is only encrypted using the shared secret, it is impossible for a third-party to tell which of the two parties generated the message.

4.4 Analysis and Discussion

Through this modeling, we have shown that a slightly modified, Tamarin-suitable version of the X3DH protocol demonstrates two of the key main promises described by Signal with its protocol: mutual authentication (which is related to cryptographic deniability) and forward secrecy.

With regards to what this says about the true X3DH protocol, we believe that this provides strong reasons to believe that the protocol which Signal implements follows the same properties as the ones we have verified. This is because we have not found any evidence that elliptic curve cryptography is particularly more or less difficult than standard Diffie-Hellman, and it is instead preferred for its efficiency [4]. Additionally, we believe that this helps support the argument that there is cryptographic deniability in X3DH; however, this remains to be formally proved in Tamarin.

5 RELATED WORK

There are some proofs of X3DH's security that already exist in the literature. Ferran van der Have [11] uses game hopping to prove that, under the Gap Diffie-Hellman assumption, the X3DH protocol is indeed secure. The author constructs a series of games, where each game slightly differs from the previous one but are all similar enough to relate to each other. Ferran van der Have is able to then relate a bound for the last game back to the first/original game.

By hopping from game to game, Ferran van der Have is able to calculate the following bound on the advantage of the adversary:

$$\mathrm{Adv}_0 \leq \frac{\binom{p+e\cdot p}{2}}{q} + p^2 s^3 \left(\frac{2e+s+1}{q} + (2e+s+1)\epsilon_{GDH}(\mathcal{A}^*) \right)$$

Here, we have that:

- Adv_0 is the advantage of the adversary.
- p is the number of parties.
- s is the maximum number of sessions.
- e is the maximum number of ephemeral/medium-term keys for each party.
- q is the group order.
- $\epsilon_{GDH}(\mathcal{A}^*)$ is the probability of guessing the correct answer for the Gap Diffie-Hellman problem.

Note that Ferran van der Have does not prove cryptographic deniability but rather only focuses on forward secrecy. Although alternate proofs exist, we believe that proving security properties of X3DH in Tamarin is still valuable because it provides greater flexibility for proving other properties or easily re-verifying the protocol if it changes at all.

6 SHORTCOMINGS AND FUTURE WORK

6.1 Elliptic Curve vs. Modular Exponentiation

As noted previously, we do not use the elliptic curve cryptography required by X3DH in our model due to Tamarin not implementing it in its solver. Here, we will briefly discuss the differences between using elliptic curves vs. modular exponentiation in Diffie-Hellman and a possible way to implement elliptic curves in Tamarin.

6.1.1 Main Differences. The cryptographic security of Diffie-Hellman relies on the difficulty of solving the discrete log problem, or inverse exponentiation, in a certain group. The

standard group that is used is the integers modulo some large prime number. It is difficult to find an integer exponent that gets you from one number in the group to another number in the group. Although there are no known polynomial-time algorithms for solving this problem, there are some approximation techniques, and even exact-solution algorithms, that reduce time complexity by significant constant factors.

Elliptic curves are simply another group where the discrete log problem is difficult to solve, and there currently are no known algorithms that perform any better than the naive solution. While this obviously provides better cryptographic security, the difficulty of solving both problems is on the same order of magnitude [9].

It is still important to note, however, that some implementation details of using one group or another may leak through even though Diffie-Hellman theoretically works the same across finite groups. However, Tamarin's solver only supports modular exponentiation, so we are unsure if our model works exactly the same as if we were to use elliptic curves.

6.1.2 Elliptic Curves in Tamarin. Tamarin uses The Maude System as its underlying rewriting logic framework which contains integrations for various SMT solvers such as Z4 or CVC4. Both of these solvers contain support for modular arithmetic but do not yet contain support for elliptic curves. Adding elliptic curves to Tamarin would involve implementing elliptic curve support in an existing SMT solver and then adding the support to Tamarin to call into the underlying solver. It may also be possible to implement elliptic curves using Maude itself by only using the primitives provided by the underlying solvers; however, this is well beyond the scope of this paper.

6.2 Exploding Solver Complexity

One of the major problems we faced during the course of this project was the quantity of computational resources and time which were required to run Tamarin on a more complicated protocol like X3DH. We model X3DH with the use of a hashing function `h` as demonstrated below on line 5 of the following block:

```
let DH1 = SPKB^~pIKA
    DH2 = IKB^~pEKA
    DH3 = SPKB^~pEKA
    DH4 = OPKB^~pEKA
    SK = h(<DH1, DH2, DH3, DH4>)
    AD = <IKA, IKB>
```

However, in practice, there were times when this created several partial deconstructions in Tamarin, which in turn heavily complicated the model in Tamarin. It seems that hashing resulted in long runtimes, as Tamarin was unable to resolve where the hashed fact came from, as described in the Tamarin manual [10]. This, in turn, led to a non-terminating sequence in Tamarin which steadily increased RAM usage until the program crashed. We were ultimately able to fix this problem by reconfiguring some of the facts and their orderings; however, these partial deconstructions heavily limited our implementation speed and complexity when they showed up for a variety of reasons, including hashing and building, throughout the project.

Even beyond the use of hashing, however, we found that Tamarin utilized substantial resources. In particular, while implementing the X3DH protocol, we initially wanted to perform four Diffie-Hellman pairings as shown in Figure 1 to create four distinct secrets which factor into the shared secret key. However, this had a long run time locally on our devices, and such a runtime made it difficult to impossible to evaluate the security promises on such a configuration of X3DH. Specifically, our key secrecy lemmas do not complete within a reasonable amount of time. On the other hand, when we removed the exponentiation with Bob's one-time key and instead ran the X3DH model with only three exponentiations (pairs 1, 2, and 3 in Figure 1), the program ran correctly. We note that the one-time key is not mandatory for X3DH as described by [6]; however, the forward secrecy properties are weakened without it.

It is unclear what causes the drastically longer runtimes; yet, we believe it may be because the addition of extra one-time keys may exponentially increase the space over which Tamarin must work to resolve lemmas. However, when we noted this, we also found that other more complex lemmas would be difficult or even impossible to implement. In particular, we found mechanisms by which other researchers have formally proved cryptographic deniability using Tamarin. Specifically, these involve running the protocol twice, once with all of the true secret keys as an honest protocol run, and another time with only public values and other values available to the attacker in the given model [1]. Given the limitations we found with the time and space used by Tamarin, we briefly attempted but ultimately did not implement formally proving deniability in Tamarin. However, as described above, the signature authentication process encoded in the X3DH protocol may intuitively help with deniability.

6.3 Double Ratchet

Another area of future work which could be explored in the future is the formal validation of Signal's double ratchet algorithm, which is a protocol for deriving new keys from an initial shared key. This allows for subsequent messages to be continually encrypted with new keys so that old messages cannot be decrypted with new ones (forward secrecy) while avoiding the need to establish a key using a full handshake protocol like X3DH for every message [5]. While the

details of this mechanism are outside the scope of this paper, we note that this is a crucial algorithm upon which the Signal app is built, and a security analysis of Signal cannot be done without considering it. As a result, future work centered around formalizing the security promises of the double ratchet algorithm (possibly in Tamarin) would also be relevant to understanding the platform as a whole.

7 CONCLUSION

As more and more people rely on messaging apps for communication, security and privacy become increasingly crucial. Messages between two parties should be kept private and unable to be decoded by an adversary. Signal is one such messaging app, and it claims that their platform achieves both forward secrecy and cryptographic deniability through the X3DH key agreement protocol.

In order to verify that the X3DH protocol does indeed guarantee forward secrecy and cryptographic deniability, we implement an approximate version of X3DH in Tamarin. Through our implementation, we formally verify that, indeed, forward secrecy and cryptographic deniability are achieved through the protocol.

REFERENCES

[1] Sofia Celi, Jonathan Hoyland, Douglas Stebila, and Thom Wiggers. 2022. A tale of two models: formal verification of KEMTLS via Tamarin. Cryptology ePrint Archive, Paper 2022/1111. https://eprint.iacr.org/2022/1111 https://eprint.iacr.org/2022/1111.

[2] Whitfield Diffie and Martin E. Hellman. 1976. New Directions in Cryptography. *IEEE Transactions on Information Theory* 22, 6 (November 1976), 644–654.

[3] Dr. Hugo Krawczyk and Pasi Eronen. 2010. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869. https://doi.org/10.17487/RFC5869

[4] Adam Langley, Mike Hamburg, and Sean Turner. 2016. Elliptic Curves for Security. RFC 7748. https://doi.org/10.17487/RFC7748

[5] Moxie Marlinspike. 2016. The Double Ratchet Algorithm. https://signal.org/docs/specifications/doubleratchet/. [Accessed 07-05-2024].

[6] Moxie Marlinspike. 2016. The X3DH Key Agreement Protocol. https://signal.org/docs/specifications/x3dh/. [Accessed 07-05-2024].

[7] Trevor Perrin. 2016. The XEdDSA and VXEdDSA Signature Schemes. https://signal.org/docs/specifications/xeddsa/. [Accessed 07-05-2024].

[8] Signal. 2024. Signal. https://signal.org/#signal. [Accessed 07-05-2024].

[9] Nick Sullivan. 2013. A (Relatively Easy To Understand) Primer on Elliptic Curve Cryptography. https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography. [Accessed 07-05-2024].

[10] "The Tamarin Team". 2024. "Tamarin-Prover Manual". https://tamarin-prover.com/manual/master/tex/tamarin-manual.pdf. [Accessed 07-05-2024].

[11] Ferran van der Have. 2022. The X3DH Protocol: A Proof of Security. https://www.cs.ru.nl/bachelors-theses/2021/Ferran_van_der_Have__4104145___The_X3DH_Protocol_-_A_Proof_of_Security.pdf. [Accessed 07-05-2024].