



Stealth Key Exchange and Confined Access to the Record Protocol Data in TLS 1.3

Marc Fischlin

Cryptoplexity

Technische Universität Darmstadt

Darmstadt, Germany

marc.fischlin@tu-darmstadt.de

ABSTRACT

We show how to embed a covert key exchange sub protocol within a regular TLS 1.3 execution, generating a stealth key in addition to the regular session keys. The idea, which has appeared in the literature before, is to use the exchanged nonces to transport another key value. Our contribution is to give a rigorous model and analysis of the security of such embedded key exchanges, requiring that the stealth key remains secure even if the regular key is under adversarial control. Specifically for our stealth version of the TLS 1.3 protocol we show that this extra key is secure in this setting under the common assumptions about the TLS protocol.

As an application of stealth key exchange we discuss sanitizable channel protocols, where a designated party can partly access and modify payload data in a channel protocol. This may be, for instance, an intrusion detection system monitoring the incoming traffic for malicious content and putting suspicious parts in quarantine. The noteworthy feature, inherited from the stealth key exchange part, is that the sender and receiver can use the extra key to still communicate securely and covertly within the sanitizable channel, e.g., by pre-encrypting confidential parts and making only dedicated parts available to the sanitizer. We discuss how such sanitizable channels can be implemented with authenticated encryption schemes like GCM or ChaChaPoly. In combination with our stealth key exchange protocol, we thus derive a full-fledged sanitizable connection protocol, including key establishment, which perfectly complies with regular TLS 1.3 traffic on the network level. We also assess the potential effectiveness of the approach for the intrusion detection system Snort.

CCS CONCEPTS

• Security and privacy → Cryptography.

KEYWORDS

Key exchange; secure channel; sanitization; TLS

ACM Reference Format:

Marc Fischlin. 2023. Stealth Key Exchange and Confined Access to the Record Protocol Data in TLS 1.3. In *Proceedings of the 2023 ACM SIGSAC*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CCS '23, November 26–30, 2023, Copenhagen, Denmark

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0050-7/23/11...\$15.00

<https://doi.org/10.1145/3576915.3623099>

Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark. ACM, New York, NY, USA, 15 pages.
<https://doi.org/10.1145/3576915.3623099>

1 INTRODUCTION

Common key exchange protocols allow two parties to agree on a session key. We investigate here the notion of *stealth* key exchange where the parties can create another joint key, called the stealth key, within an execution. The steps to generate this extra key are embedded covertly into the regular execution, such that outsiders remain oblivious if the stealth mode has been used or not. The derived stealth key should be secure even if an adversarial parties gets to know—or can even control—the regularly established session key in such an execution.

1.1 The Approach

The idea of building stealth key exchange protocols relies on the widespread deployment of nonces in key exchange protocols, e.g., in TLS 1.3 each party sends a random 256-bit value in the Hello messages. In terms of security these nonces should ensure that sessions are unique, but usually one does not need to assume anything beyond this property. One can thus use these nonce values to embed further useful information which can be used to derive the additional stealth key. This idea already appears in several anti-censorship protocols like Telex [60] and Decoy Routing [34].¹

Let us concretely consider the TLS 1.3 key exchange in the (EC)DHE mode with server authentication. For a simplified version of this protocol see (the left part of) Figure 1. Besides the client and server nonces N_C and N_S , the parties also run a Diffie-Hellman key exchange protocol (over an elliptic curve), deriving a joint secret g^{xy} from the parties' shares g^x and g^y . This secret g^{xy} is then used in the key derivation step, applied to nonces N_C , N_S and additional information.

The idea is now to embed suitable elliptic curve points for yet another Diffie-Hellman key exchange in the TLS 1.3 nonces. Specifically, both parties on top generate another Diffie-Hellman key g^{ab} by embedding their corresponding shares g^a and g^b into the nonces N_C and N_S . The stealth key is then computed as in the original protocol, but by swapping the role of the Diffie-Hellman values and the nonces. That is, the stealth key is now derived via the key derivation function, applied to the secret g^{ab} for “nonces” g^x and g^y (and the other data). See the right part of Figure 1.

The final step is to make sure that the embedding of the extra Diffie-Hellman values remains hidden. Here we use the Elligator

¹We give a comprehensive comparison to related works in Section 2, after having discussed the main ideas.

proposal of Bernstein et al. [8] which allows to efficiently create elliptic curve points which are statistically indistinguishable from uniform bit strings. We discuss the exact embedding algorithms within. Once this is accomplished, we have implemented the stealthiness.

1.2 Applications

We briefly discuss here some applications of stealth key exchange. The applications partly also serve as a motivation for our security model in the next subsection. We remark once more that we discuss related concepts in more detail in Section 2.

The first application is a weak form of deniable communication. Since the stealth mode cannot be distinguished from an actual key exchange execution, an outsider cannot know if the parties have agreed on the extra key. If now the parties use this key to encrypt the communication upfront, before pushing it through the channel secured by the actual session key, then they can claim to have sent random data.

Another application is to reduce the trust in Trusted Execution Environments (TEEs). Such environments nowadays usually support, among others, the generation, storage, and computation of Diffie-Hellman keys. Hence, when used within a protocol like TLS, the user may hope to benefit from the additional security guarantees of the TEE (albeit the security of such TEEs may be weaker than expected, e.g., see [11] for a discussion for TrustZone, for instance). When using the TEE, however, the user remains oblivious about how the Diffie-Hellman keys are generated or stored within the TEE, or even if they are leaked, opening up the possibility of key escrow. With a stealth key exchange the users may generate the additional stealth key and pre-encrypt transmitted data under that key. In this sense the user profits from a “good” TEEs and, at the same time, reduces the risk for a “bad” TEE.²

A third example where stealth key exchange may be useful is malware detection for TLS-encrypted communication. So far, one had to share the session key with the middlebox (e.g., an intrusion detection system) in order to allow scanning for potential threats. The sharing of ad-hoc session keys is a challenging problem in itself, but so far practical approaches follow an all-or-nothing principle: either the middlebox has access to the entire communication or no access at all. The stealth key exchange would allow the users to use the extra stealth key to pre-encrypt parts of the communication and only give the middlebox access to other parts. And it remains up to the users to decide which parts remain end-to-end encrypted. We note that the actual implementation of this idea is far from trivial and presented in brief in Section 6 and more comprehensively in the full version [26], with a closer look at the potential integration into the intrusion detection system Snort in Section 7.

1.3 Security of Stealth Key Exchange

We next discuss what kind of security properties we expect from stealth key exchange protocols. This has previously not been captured rigorously, according to common security models for key exchange. Intuitively, there are two relevant properties. First, we demand that the stealth key is confidential, even if the session key

derived in the same execution is disclosed, or, following the TEE application example, even if the adversary gets to influence the session key. Confidentiality of the stealth key here refers to the common notion of indistinguishability from random. We also assume, vice versa, that the session key remains secure if the stealth key is revealed. The second property of a stealth key exchange is that one cannot tell apart executions in which a stealth key is generated from those running merely a regular key exchange execution. This hides the fact whether a stealth key has been agreed upon or not.

We give a security definition in the game-based style of Bellare and Rogaway [5], capturing potential correlations between the session key, the stealth key, and the stealthiness of the execution. That is, classical key exchange protocols define confidentiality of the session key via a secret challenge bit b , determining if the adversary gets to learn the session key ($b = 0$) or a random string instead ($b = 1$) in a challenge. The task of the adversary is to predict the bit b . For our security definition we use the same challenge bit b to seize all secrecy properties simultaneously: If the bit b is 0 then we let the parties run in stealth mode, and also hand the adversary the session key resp. the stealth key if requested. If, on the other hand, the bit b is 1, then the parties run in regular mode, and if the adversary asks to be challenged about a key then we return a random (session or stealth) key instead.

We note that the security model with its session-centric view of one party’s communication in an execution introduces some interesting effect for the stealthiness. That is, a party may start in stealth mode, with the goal of establishing another key, whereas the intended communication partner does not and instead runs in regular mode. Unless the two parties have already established a shared secret before, they cannot secretly coordinate if they both want to run in stealth mode when the key exchange begins. They can learn this after completion of the key exchange protocol, of course, for example, by trying to use the extra key.

1.4 Stealth TLS 1.3

We next prove that the stealth version of TLS 1.3 satisfies the strong security guarantees of a stealth key exchange protocol, when using an appropriate embedding. For the nonce embeddings we use Elligator 2 for Curve25519 [8], since Curve25519 is also one of the recommended elliptic curves for TLS 1.3. Hence, our security proof shows that the Elligator embedding allows to derive a stealth key which is as secure as the regular TLS channel key (for Curve25519), and remains secure if the session key is leaked or even determined by the adversary. In other words, deriving another fresh key within a given TLS 1.3 is possible, and the fact that this extra key is derived cannot be spotted from the outside.

One could argue that stealth key exchange does not improve over the trivial solution to run another execution with the partner, say, another TLS 1.3 exchange, to generate yet another key. However, such extra executions may be easy to detect and may be prohibited, e.g., for political reasons. The stealth key exchange mode, on the other hand, goes undetected. Another difference lies in the availability of the secret to authorized parties. If a government enforces key escrow for any connection, then simply running two instances would not allow to create a key shared only by the communication

²Note that our approach uses the random nonces to establish the stealth key. Hence, at least the nonce generation cannot be outsourced to the TEE and its random generator.

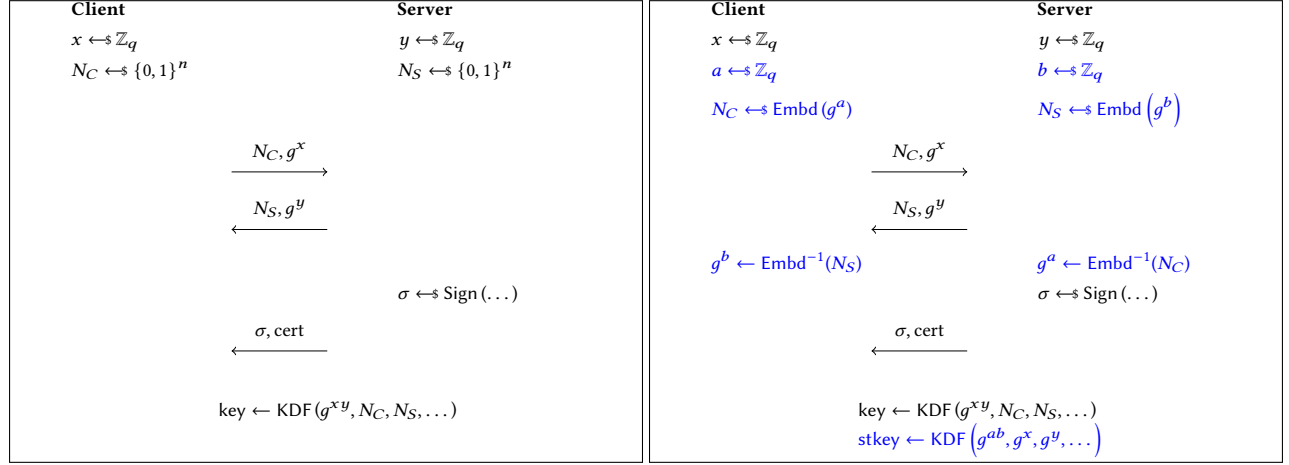


Figure 1: Simplified version of TLS 1.3 (left) and stealth mode (right).

partners. We note that our intrusion detection system case displays an example where (partial) access to the data may be desired. Remarkably, the embedding technique can be used to provide such a trade-off. Finally, and this depends on the embedding and the protocol, the stealth mode may be faster than two executions, especially in terms of latency.

1.5 Sanitizable Channel Protocols

As a concrete application of the stealth version of TLS 1.3 we show how to lift the mode to accomplish (controlled) sanitization for a TLS 1.3 channel. Going back to the intrusion detection example, we let the sender and receiver run the stealth key exchange protocol, agreeing on the stealth key for establishing an end-to-end protected connection, and letting the receiver use a static Diffie-Hellman part shared with the detection system. The latter implies that detection system, also called sanitizer, and receiver and sender all share the regular session key of the connection. We note that the static Diffie-Hellman share demolishes forward secrecy of the regular key, but our security proof shows that the stealth key is nonetheless forward secure.

The idea is now to let the sender and receiver use the stealth key to conceal information from the sanitizer, protecting the inner data m_{sec} with the stealth key to derive an inner ciphertext c_{sec} . Then the sender inserts this inner ciphertext c_{sec} together with the accessible part m_{plain} of the message through the regular channel protocol for the session key. This allows the sanitizer to check for the plain part only, hiding the m_{sec} -part from the sanitizer. In fact, we use a more fine-grained distinction into secure, confidential, authenticated, and plain message parts.

We show the above approach is secure if the underlying authenticated encryption scheme is secure, in a suitable model for sanitizable channels. We emphasize that the final ciphertexts are slightly longer than if encrypting m_{sec} and m_{plain} directly, because the inner ciphertext c_{sec} also includes an authentication tag. Nonetheless, when using either of the two suggested authenticated encryption methods in TLS 1.3, GCM or ChaChaPoly, the final ciphertext is a legitimate ciphertext according to TLS 1.3 standards. Thus, when

executing the stealth key exchange protocol together with the sanitizable channel, this perfectly complies with the TLS 1.3 standard on the network level.

An interesting feature for the sanitizable channel protocol is that we can preserve the stealthiness from the key exchange to the channel protocol. This means that even the sanitizer cannot know if the sender and receiver actually exchange additional messages in the inner ciphertext. For this we use a common property of the authenticated encryption schemes, namely, that one cannot distinguish actual ciphertexts created with the secret key from random bits. Our security model will capture this stealth property of the sanitizable channel, such that our TLS 1.3 based solution, shown secure in this model, also provides this extra feature.

We finally discuss how our sanitizable channel protocol could be integrated into a network intrusion detection system like the open-source program Snort. Snort comes with a predefined set of rules for checking network traffic, of which roughly half touch upon HTTP traffic. Suppose we grant Snort, as the sanitizer, access to HTTP meta-information like the header data by putting these data in the accessible part m_{plain} , but hide the actual HTTP content from Snort in the inner ciphertext c_{sec} . Then we can still cover a vast majority of all HTTP-related rules in Snort but now work over encrypted communication.

2 RELATED WORK

Our result relies on several ideas and techniques appearing in the literature. We discuss here —and delineate from— the most relevant works.

2.1 Steganography

Stealth key exchange is related to steganographic techniques in cryptography which can be traced back to Simmons' work about the prisoner's problem [53]. The case of public-key steganography has been studied extensively, starting with the initial idea mentioned in [1, 12] to the first formalization by von Ahn and Hopper [56]. Several other works focusing on steganographic techniques for public-key encryption followed, e.g., [4, 6, 32, 39]. We note that

only the work by von Ahn and Hopper [56] discusses key exchange but merely for passive adversaries; all other works in this realm consider encryption.

The most important difference to our setting here is that steganographic schemes embed a message into a regular communication, whereas stealth key exchange “only” aims to generate an extra key. This may sound like a subtle difference but has crucial consequences for the design. Steganographic schemes often embed bits of the message via rejection sampling [4], such that for transmitting each bit covertly many samples and one ciphertext are necessary. In fact, Dedic et al. [20] show that an exponential number of samples is required unless one exploits specifics of the communication channel. We can bypass the lower bounds since we are only interested in the partners agreeing on an additional secret key.

2.2 Embeddings

The idea of embedding elliptic curve points as bit strings in an indistinguishable way dates back to Möller [43]. In his solution, he uses the fact that the x -coordinate of the point either denotes a valid curve point or a valid point on the twist. This allows to represent public keys as random strings. Möller’s idea has been used in StegoTorus [58] to include steganographic techniques in TOR.

The most widely used embedding today is the Elligator approach of Bernstein et al. [8]. It comes in two flavors, Elligator 1 and Elligator 2. Elligator 1 is based on an approach by Fouque et al. [28] and works for some elliptic curves. Elligator 2 is more general and in particular works with Curve25519 one of the options in TLS 1.3 for elliptic curves. This is why we use Elligator 2 here. We also remark that Bernstein et al. [8] discuss issues with the coventness of other elliptic curves available in TLS 1.3, especially with NIST’s curve P-256 which may not easily yield almost uniform bit strings. The reason is basically that the order of curve P-256 is not sufficiently close to a power of 2.

Tibouchi [55] presents an improvement for Elligator, denoted as Elligator Squared, which overcomes the issue of repeated sampling to find a suitable curve point and may thus be less vulnerable against time-based side channels. Aranha et al. [2] further improve the efficiency for Elligator Squared. Unfortunately, the size of the embedded bit string in Elligator Squared is twice as large as in the Elligator case, such that we could not embed it easily into the 256-bit nonce in TLS 1.3 for the same security level. That is, we had to use a 128-bit curve instead, which provided at most 64 bits of security.

2.3 Analyses of TLS 1.3

TLS 1.3 [50] has been developed between 2014 and 2018 by the IETF. The process has been accompanied by a number of scientific analyses during the standardization, both cryptographically [23, 36, 37] as well as by formal methods and symbolic analyses [9, 10, 13, 14, 21]. The most relevant analysis for us here is the one in [23] (see also [24] for an updated version) as it uses a similar security model (but in the multi-stage setting). Noteworthy, since we give a reduction to the security of the basic TLS 1.3 protocol, the latest results about tight security proofs of TLS 1.3 [17, 22] immediately

transfer to our setting. Note that we do not investigate the pre-shared key mode of TLS 1.3 such that corresponding tightness results as in [16] do not apply to our setting here.

For the sanitizable channel protocol we use that GCM is a secure authenticated encryption scheme with associated data (AEAD) when used with a pseudorandom permutation [42] like AES in TLS 1.3. The same holds for the composition of ChaCha20 and poly1305 [48], assuming ChaCha20 is pseudorandom and poly1305 is a universal hash function. In our security proof we use additional common properties of such AEAD schemes, namely, that ciphertexts cannot be distinguished from random and that the length of the ciphertext can be deduced from the length of the input message. Both AEAD schemes used in TLS 1.3 satisfy these properties (under the aforementioned assumptions).

2.4 Middleboxes

It is well known that end-to-end encrypted payload and packet inspection by middleboxes are usually irreconcilable. Clearly, the privacy requirements of the users are very important. However, De Carné de Carnavalet and van Oorschot [19] give an overview over cases where accessing secured data may still be desirable. This includes legal reasons like lawful interception or fraud detection, security reasons like malware download protection or intrusion detection, performance reasons like caching and compression, and other reasons like parental control. Note that some cases are even in the interest of the end users.

A simple solution is to make sure that the middlebox has access to the channel key such it can access the payload in clear. In previous TLS versions this could be implemented relatively smoothly by using static keys in the key exchange, for which the middlebox knows the secret keys. But this, of course, sacrifices forward security and was one of the reasons to forgo this option in TLS 1.3. Nonetheless, Green et al. [30] describe a TLS 1.3 variant with static keys to resurrect accessibility, at the cost of forward secrecy.

Another option is to split the end-to-end connection into two connections, one from each user to the middlebox. However, De Carné de Carnavalet and Mannan [18] point out potential vulnerabilities due to sloppy certificate checks of middleboxes. Other potential vulnerabilities are unwanted modifications of the content or defaulting to weak cryptography due to the middleboxes. The middlebox-aware TLS protocol maTLS [40] attenuates this by introducing auditable middleboxes, yet still breaking end-to-end security.

More sophisticated alternatives for the middlebox problem are the BlindBox protocol [52] and the recently proposed concept of zero-knowledge middleboxes (ZKMB) [31]. In the (most basic version of the) BlindBox protocol the sender sends encrypted tokens in addition to the protected communication, secured under a token key derived also from the channel key. The middlebox holds a (secret) set of detection rules in form of keywords. The client provides the middlebox at the beginning of the connection with the encrypted versions of the keywords such that detection is possible. This is done obviously, without revealing the token key. The overhead of the cryptographic operations make BlindBox an order of magnitude slower than original connections.

As pointed out by the authors of the ZKMB solution [31] the issue with BlindBox is that it modifies the actual connection protocol. Preserving the protocol structure is an important compatibility property. The ZKMB protocol overcomes this limitation for showing policy compliance. The idea is that the client and server establish a regular connection, and the client proves in zero-knowledge to the middlebox that the encrypted payload obeys certain rules. Hence, the client-server connection entirely runs the original connection protocol. Relying on recent progress in efficient zero-knowledge proofs the overhead for long-lived connections is only a few milliseconds. For regular TLS connections the overhead in terms of time and storage for precomputations is still significant, though.

Our stealth TLS 1.3 variant comes close in spirit to the multi-context TLS (mcTLS) solution [44]. In mcTLS the parties generate an end-to-end TLS connection but, at the same time, each party also establishes a connection with the middlebox. This results in different symmetric keys, one shared between the end points, and one shared between each party and the middle box. The different keys can now be used to protect the payload in such a way that the middlebox is able to access data encrypted with the key shared with the sending party, called context encryption in [44].

Our solution for middleboxes follows the same idea of using context encryption, but has several advantages. First, our solution does not need to modify the TLS 1.3 protocol on the outer layer; only the pre-encryption the inner data increases the length of the outer encryption (which remains a valid channel encryption). This is an important compatibility property accomplished with the ZKMB protocol [31]. Second, and related to the necessary but not necessarily sufficient compatibility property, we achieve stealthiness (almost) for free. Third, mcTLS puts additional trust in the middlebox in terms of certificate verifications.

Finally, let us remark that the BlindBox solution and especially the ZKMB protocol have an advantage in terms of flexibility and security over our approach. Both protocols support checking of general properties which are hidden from the middlebox. In contrast, our solution only allows for context encryption, dividing the payload coarsely into visible and protected parts. In addition, the deployment of the (semi)-static keys diminishes forward secrecy. In return, our solution blends in easily into the existing protocol and does not require any modifications on the network layer.

2.5 Anti-censorship

Closely related to the issue of middleboxes in secure connections is the question of anti-censorship. The idea of using covert data to prevent censorship has been put forward in several works before, and some approaches share some of the techniques used here. Arguably, the most prominent examples in this regard are Telex [60], Cirripede [33], and Decoy Routing [34]. All three approaches are based on similar principles, but differ in details. The idea is to have a client in a TLS connection covertly trigger a dedicated decoy server on the path to the actual server. This allows to bypass censorship since the decoy server, once alerted, will contact the server on behalf of the client and relay the communication. In order to do so, the client and the decoy server need to establish a joint secret which the client uses in the connection to the actual server and which is thus known to the decoy server. The approaches differ

in the way how the decoy server is triggered and how the joint secret between client and decoy server is computed.

Both Telex and Decoy Routing let the client embed a secret tag into nonce in a TLS connection. Specifically, the client holds a public key g^s of the decoy server and embeds $g^r | H(g^{rs})$ in the nonce for randomness r , hash function H , and a (short) Diffie-Hellman key g^{rs} . The decoy server is able to detect that the second half equals the hash while outsiders should not be able to distinguish the cases. The client is then supposed to use $KDF(g^{rs})$ as the secret in the key establishment with the server, such that the decoy server also holds the session key. We note that the follow-up design of TapDance [59] explicitly uses Elligator 2 for the embedding.

Decoy Routing [34] also uses the nonce in the client hello message to trigger a special event, but relies on a pre-shared secret between client and station to embed the tag via HMAC. It also uses this pre-shared secret to agree on the client's secret in the connection. On the other hand, Cirripede [33] once more uses the Diffie-Hellman based approach, but uses a pre-shared secret during registration to ensure that client and decoy server know the same connection secret.

The main difference to our work here is that all the aforementioned approaches are mainly interested in the covertness to bypass censorship. In contrast, we are interested in the (combined) stealthiness and key secrecy in an end-to-end connection, albeit our application examples show that third parties can get involved if desired. Another difference is that we provide a rigorous cryptographic analysis of the achieved properties. The final point is that we work with TLS 1.3 whereas the earlier proposals of course considered earlier versions.

2.6 Anamorphic Encryption

Recently, Persiano et al. [47] introduced the notion of anamorphic public-key encryption. The idea is to allow the sender and receiver covertly transmit information, even if an observer gets to determine the message to be sent, and gets access to the secret key of the recipient. Their approach is to have an additional indistinguishable key generation algorithm which, on top of the public and secret key, outputs another special key, the double key. When sharing this double key with the sender, the two parties can covertly communicate. Persiano et al. [47] give constructions based on rejection sampling and based on the Naor-Yung paradigm. In [38] the idea was extended to signature schemes.

Anamorphic encryption, like the approach of embedding information into nonces, displays similar ideas to covertly communicate in the presence of observers. There are, nonetheless, major differences between our work and anamorphic encryption. At foremost, we work in the domain of key exchange, implicitly solving the question on how the stealth (or, double) key is securely shared between sender and receiver. Then, our solution even works in the setting where the observer chooses the ephemeral secrets on the receiver's side (cf. the TEE example), whereas in anamorphic public-key encryption the receiver presents a suitable secret key to the observer. A disadvantage of our solution is that, when referring to communication of data, our embedding of the covertly sent messages in the channel protocol increases the length of the ciphertext, such that we can hardly hide the fact that we are using a scheme with allows

for covert communication. In contrast, in anamorphic encryption the “anomorphic” ciphertexts are indistinguishable from the ones of a given innocuous system.

2.7 Sanitizable Cryptography

The notion of sanitizable signature schemes has been introduced by Ateniese et al. [3]. Such schemes allow a designated party, called the sanitizer, to modify a signed message according to some predefined rule, such that authenticity of the derived message is still verifiable. We lift here this idea to channel protocols. As the intrusion detection system in our setting plays the role of the designated party being able to make admissible changes to the payload, we use the term sanitizable channel here.

Many works in the area of sanitizable cryptography nowadays focus on signature schemes, with only a few exceptions. One is the work by Fehr and Fischlin [25] which covers sanitizable signature schemes. Such schemes combine (public-key) encryption with signatures, making sure that the sanitizer does not learn the original message when sanitizing the signature, nor possibly even the resulting sanitized message. The work does not investigate symmetric-key channel protocols.

Access control encryption, introduced by Damgård et al. [15] and subsequently extended by [29, 35, 57], also involves a sanitizer which ensures that only admissible information can be passed from senders to receivers. Access control encryption rather implements the classical access control requirements (like the no-read rule and the no-write rule) and moreover aims to provide anonymity. All of the aforementioned solutions are geared towards public-key cryptography and indeed use public-key operations to achieve the security properties. Neither of the works looks into real-world channel protocols with a single sender and receiver sharing a symmetric key.

2.8 Other Notions of Stealth Key Exchange

The term “stealth” has been used in connection with key exchange before, yet with different meanings. Rafat [49] explicitly used the term stealth key exchange in order to describe a (plain) Diffie-Hellman key exchange executed over a seemingly covert channel, such as a frequently changing web site. In a sense, this means to execute a key exchange protocol over a steganographic communication channel. Patgiri and Muppalaneni [46] propose an (unauthenticated) key exchange protocol, called Stealth, which runs four Diffie-Hellman key exchanges and uses these keys to encrypt messages in a nested but unauthenticated form. Neither of the proposals aims at embedding another key in an existing key exchange protocol execution, nor provides a formal security analysis.

3 SECURITY MODEL FOR STEALTH KEY EXCHANGE

We start by presenting the security model for stealth key exchange. We follow the classical game-based model of Bellare and Rogaway [5]. We only consider the single-stage setting where the parties agree on a single session key upon termination of the key exchange phase. TLS 1.3, in contrast, is a so-called multi-stage protocol [27] in which several keys are derived —and possibly deployed— during the key exchange phase.

We assume that we are given a two-party key exchange protocol Π . The protocol should be correct in the sense that, if two parties faithfully execute the protocol then they derive the same session key. We capture this more liberally by demanding that in such an execution the two parties output the same session identifier sid which identifies connected sessions. The choice of sid is part of the protocol description. We will later stipulate as a security requirement that identical session identifiers sid also imply identical session keys.

3.1 Attack Model

We assume a set of user identities \mathcal{U} , each user u being equipped with a key pair $(\text{sk}_u, \text{pk}_u)$ generated at the outset of the attack, together with a valid certificate cert_u containing the public key pk_u . We assume that algorithm KGen is used to create each certified key pair. The certificates and thus also the public keys are known to the adversary. Let C be an initially empty set of corrupt users. If the adversary later corrupts a user $\text{id} \in \mathcal{U}$ then id is added to C . We note in the initialization of a session we allow a party’s identity to be set to $*$, indicating that this party does not authenticate towards the other party. The understanding here is that $*$ matches any entry from \mathcal{U} , i.e., $\text{id} = *$ for any $\text{id} \in \mathcal{U}$ and also $* = *$.

There is also a global bit b for defining security, chosen randomly at the outset and hidden from the adversary. This bit determines if the adversary gets to see the actual (session or stealth) key or a random value. Here, we assume that the session key and the stealth key are chosen according to some efficient distributions $\mathcal{D}_{\text{regular}}$ resp. $\mathcal{D}_{\text{stealth}}$. The bit also decides if to run in stealth or regular mode, for sessions where the adversary does not explicitly determine the choice.

Sessions capture the state of a communicating party within the key exchange protocols. They are described by a tuple

(label, owner, party, partner, role, mode, state, sid,
key, stkey, isTested, isRevealed, isCorrPrtner),

where label is a unique administrative identifier, owner is a user identity, party and partner are the user identities indicating the intended communication partners (with $\text{party} \in \{\text{owner}, *\}$, where $\text{party} = *$ or $\text{partner} = *$ denotes that the party does not authenticate), $\text{role} \in \{\text{initiator}, \text{responder}\}$ describes the role of the session, $\text{mode} \in \{\text{regular}, \text{stealth}\}$ describes the mode, $\text{state} \in \{\text{accept}, \text{reject}, \text{running}\}$ the status of the execution, sid the session identifier (initialized to \perp and set upon acceptance), key the session key (initialized to \perp and set upon acceptance), stkey the stealth key (initialized to \perp and set upon acceptance in mode stealth), Boolean values isTested and isRevealed (with sub types regular and stealth, all four entries set to false in the beginning), and Boolean value isCorrPrtner initialized to false. We sometimes write label.owner , label.partner etc. for the corresponding entries in the tuple for the unique identifier label.

The adversary can communicate with each session and change its status through oracle queries. We highlight here two important aspects related to the stealthiness. One is that the adversary can, upon initializing a session, determine the mode, i.e., if the session should execute a regular protocol execution or run in stealth mode. But we also allow the adversary to leave this entry unspecified,

in which case we assign the mode according to the challenge bit b . We then need to prevent trivial attacks in which the adversary checks (via Test or Reveal queries) if there exists a stealth key or not, thereby learning the secret bit b .

The other important point refers to the independence of the stealth key from the session key. Since we want the stealth key to be confidential even if the adversary has control over the cryptographic secrets for the regular key exchange part (cf. the TEE example), we also admit the adversary to optionally provide the ephemeral and long-term secrets upon session initialization. If the adversary chooses to do so, then the session key is marked as revealed, but the stealth key can still be tested. We can also view this as a possibility to disclose the secrets for deniability reasons, but still be able to use the stealth key securely. Like session identifiers the precise definition of this auxiliary data is part of the protocol description, potentially also causing the protocol to abort immediately if aux is not sound.

Init (owner, party, partner, role, [mode], [aux]): Initializes a session for user owner $\in \mathcal{U}$, with party $\in \{\text{owner}, *\}$, with intended partner partner $\in \mathcal{U} \cup \{*\}$, role, and if the optional argument mode is presented, in the corresponding mode. If no mode is determined then we use mode \leftarrow regular if $b = 0$ and mode \leftarrow stealth if $b = 1$. In this case, i.e., if no mode argument is passed on, we also set isRevealed.stealth \leftarrow true; else we still let isRevealed.stealth \leftarrow false. This is to prevent trivial attacks on the bit b by testing for the existence of a stealth key if no mode value is given.

Also set state \leftarrow running, sid \leftarrow key \leftarrow stkey $\leftarrow \perp$ and isTested.regular \leftarrow isTested.stealth \leftarrow isRevealed.regular \leftarrow false. If partner $\in \mathcal{C}$ is corrupt then mark the entry isCorrPrtnr \leftarrow true, else isCorrPrtnr \leftarrow false. Generate a new identifier label and store the passed values in the corresponding entries of the tuple. If the optional argument aux is present then the party will use this value in the regular session as auxiliary input, but we set isRevealed.regular \leftarrow true; if no value aux is passed then the party follows the protocol description. Returns label to the adversary.

Send (label, m): Sends protocol message m to the session with label. Here, m may be empty if the session owner is the initiator and should start sending the first message. If the session label accepts when processing the incoming message and changes to state state \leftarrow accept, then label.sid must be set according to the protocol description to a value different from \perp . In this case, the session must also set a session key label.key and, if run in stealth mode, label.mode = stealth, also a stealth key label.stkey.

Corrupt (id): Takes as input a user identity id and returns sk_{id} . Sets in all running sessions label.state = running with this intended partner label.partner = id the corruption entry label.isCorrPrtnr \leftarrow true. Note that completed sessions are not affected, in order to implement forward secrecy.

Reveal (label, mode): Takes as input a session label and a requested mode. If the session has not accepted, label.state \neq accept, or has been revealed before, label.isRevealed.mode = true, then immediately return \perp . Else, if the adversary wants to learn the session key, mode = regular, then return key

and set label.isRevealed.regular \leftarrow true. If the adversary requests the stealth key, mode = stealth, and the session has been run in stealth mode, label.mode = stealth, then return the stealth key stkey and set label.isRevealed.stealth \leftarrow true. In any other case return \perp .

Test (label, mode): Takes as input a session label and a requested mode. If the key has been tested before, label.isTested.mode = true, or the session has not accepted, label.state \neq accept, then immediately return \perp . Else, if $b = 1$ then return the session key key (if mode = regular) resp. the stealth key stkey (if mode = stealth), where potentially stkey = \perp . If $b = 0$, on the other hand, pick a random key $k \leftarrow \mathcal{D}_{\text{mode}}$ and return k . In either case, $b = 0$ or $b = 1$, set label.isTested.mode \leftarrow true.

We assume that the adversary eventually stops and outputs a guess b^* for b . We denote by $\text{Exp}_{\mathcal{A}, \Pi, \text{KGen}, \mathcal{U}}^{\text{StKey}}$ the above experiment of adversary \mathcal{A} against the key exchange protocol Π , in which one first creates the certified keys for the users in \mathcal{U} via algorithm KGen, and picks a challenge bit $b \leftarrow \{0, 1\}$, and then lets the adversary interact with the oracles as specified above.

3.2 Security Requirements

We follow the common security notions for session matching and key secrecy. The matching property says that identical session identifiers imply identical keys. Note that for stealth keys this can only hold if both parties were running in stealth mode. Uniqueness refers to the fact that at most two sessions should be partnered. The opposite role property states that in two partnered sessions one party takes the role of the initiator and the other party the role of the responder. Authentication says that partnered sessions point to the same intended partner. Note that here we use that $*$ matches any identity from \mathcal{U} (and $*$ itself) by definition, such that unauthenticated parties always obey this property. We remark that our session matching coincides with the notion in [24] when considering only single-stage security for the final keys.

Definition 3.1 (Session Matching). Let Π be a stealth key exchange protocol for users \mathcal{U} and key generation algorithm KGen, and \mathcal{A} be an adversary. Consider experiment $\text{Exp}_{\mathcal{A}, \Pi, \text{KGen}, \mathcal{U}}^{\text{StKey}}$ as above. Let $\text{Exp}_{\mathcal{A}, \Pi, \text{KGen}, \mathcal{U}}^{\text{Match}}$ denote the event that any of the four following properties is violated during the execution of the experiment:

Matching Keys: For any accepting sessions label, label' with label.sid = label'.sid $\neq \perp$ we have label.key = label'.key' $\neq \perp$ and, furthermore, if label.mode = label'.mode = stealth, then also label.stkey = label'.stkey $\neq \perp$.

Uniqueness: There do not exist three distinct accepting sessions label, label', label'' such that label.sid = label'.sid = label''.sid $\neq \perp$.

Opposite Roles: There do not exist distinct accepting sessions label, label' such that label.sid = label'.sid $\neq \perp$ but label.role = label'.role.

Authentication: For any distinct accepting sessions label, label' with label.sid = label'.sid $\neq \perp$ we have that label.party = label'.party as well as label.partner = label'.partner.

For the common asymptotic security notions we demand that for any efficient adversary \mathcal{A} the probability of $\text{Exp}_{\mathcal{A}, \Pi, \text{KGen}, \mathcal{U}}^{\text{Match}}$ is negligible.

Since we subsume both key secrecy and the indistinguishability of regular and stealth executions under one notion, we rather call the combined property indistinguishability. This property says that the adversary cannot predict the challenge bit b significantly better than guessing. For this, we need to exclude some trivial attacks, though. The first two properties say that a tested key in a session cannot be revealed, and that the tested key cannot be revealed or tested in a partnered session. Recall that excluding testing on both sides is usually an admissible strategy, since the adversary can already deduce the response for the second test itself, as partnering is usually publicly verifiable.

The third property captures cases where the adversary could already know a tested key trivially. This can either be because the partner is not authenticated (partner = *) or if the partner has been corrupted before the session has been completed (isCorrPrtner = true). Recall that, if the adversary corrupts the partner of a session after completion, then the isCorrPrtner predicate is not set. This ensures forward secrecy. To strengthen the notion, we even allow corrupt or unauthenticated partners if the session has been involved in a genuine execution run exclusively by the honest instance of the partner, i.e., if there is another session label' partnered with the tested session.

Definition 3.2 (Indistinguishability). Let Π be a stealth key exchange protocol for users \mathcal{U} and key generation algorithm KGen , and \mathcal{A} be an adversary. Consider experiment $\text{Exp}_{\mathcal{A}, \Pi, \text{KGen}, \mathcal{U}}^{\text{StKey}}$ as above. The adversary \mathcal{A} wins the experiment $\text{Exp}_{\mathcal{A}, \Pi, \text{KGen}, \mathcal{U}}^{\text{StKey}}$ denoted as event $\text{Exp}_{\mathcal{A}, \Pi, \text{KGen}, \mathcal{U}}^{\text{Ind}}$ being equal to 1, if $b^* = b$ and, in addition, all the following points are satisfied:

No Reveal nor Test for the same key: For any accepting session label and any mode $\in \{\text{regular}, \text{stealth}\}$, if we have label.isTested.mode = true then label.isRevealed.mode = false.

No Reveal nor Test on partner for tested key: For any accepting session label and any mode $\in \{\text{regular}, \text{stealth}\}$ with label.isTested.mode = true there does not exist a session label' \neq label with label.sid = label'.sid such that label'.isRevealed.mode = true or label'.isTested.mode = true (or both).

No tested key with unauthenticated or corrupt partner (unless there is a matching honest session): For any accepting session label and any mode $\in \{\text{regular}, \text{stealth}\}$ such that label.isTested.mode = true, either label.partner \neq * and label.isCorrPrtner = false, or there exists an accepting session label' \neq label with label.sid = label'.sid.

In the usual asymptotic notation we would now demand that the protocol Π provides indistinguishability (for \mathcal{U} and KGen) if for any efficient adversary the probability of $\text{Exp}_{\mathcal{A}, \Pi, \text{KGen}, \mathcal{U}}^{\text{Ind}}$ returning 1 is at most negligibly above $\frac{1}{2}$.

4 STEALTH TLS VERSION

We next describe our stealth version of TLS 1.3, called sTealS, and prove it to be secure. For this we assume that the client and server use an elliptic curve for the Diffie-Hellman steps which supports efficient embeddings. As a concrete example, the parties may use Curve25519 with Elligator 2 as explained in Section 4.2. The security proof appears in the full version [26].

4.1 Protocol Description

We describe here the the case of both parties running either in regular or in stealth mode. If only one party runs in stealth mode it still tries to compute the stealth key as described within, and will succeed with overwhelming probability to compute another key—although the other party does not hold the stealth key.

The protocol follows the idea outlined in the introduction. In regular mode it executes a (EC)DHE-variant of the TLS 1.3 protocol with optional authentication of the parties. The protocol starts with the parties computing the early secrets (key_{bind}, key_{cats}, key_{eems}) from the pre-shared key (preset to 0 for the (EC)DHE case). Since we are only interested in the stealthiness of the final traffic application keys (for client and server), denoted as key_{cats} and key_{sats} in the protocol, we assume that all intermediate keys are made immediately available to the adversary (which can be formally implemented in multi-stage settings via a Reveal query).

The actual protocol execution start with the client sending a client hello message CH, which includes a 256-bit nonce N_C , and a client key share CKS carrying a Diffie-Hellman contribution g^x . In the regular mode the client picks the nonce N_C randomly, whereas in stealth mode N_C is the embedding of another Diffie-Hellman share g^a . We note that some mild restrictions on g^a apply, i.e., it must be suitable for the embedding (see Section 4.2). We write $a \leftarrow E_q$ for the sampling according to this restriction. The server answers accordingly with the server hello SH and (random or embedded) Nonce N_S and server key share SKS with value g^y . We remark that, formally, the key share messages are part of the hello messages but it is convenient for us to make them explicit. We also require that Diffie-Hellman shares like g^x and g^y can be represented with 256 bits, as is the case for example for Curve25519.

The authentication is done via signatures σ_C on the client side resp. σ_S on the server side for the data exchanged so far. When sending this signature in the client certificate verify message CCV the client also includes the certificate in the CCERT message. Analogously for the server (which goes first to save a round trip). We note that we assume that the other party checks the signature and the certificate, and also that the certificate identity matches the pre-specified peer identity. These messages are protected under the handshake traffic secrets of the client (key_{chts}) and server (key_{shts}), respectively. Once more we assume that these intermediate keys are handed to the adversary, such that we can in particular decrypt the actually exchanged protocol messages.

The parties also use message authentication keys key_{CFIN} and key_{SFIN} to compute a MAC over the communication data. Unlike the signature step this part is mandatory. However, remarkably it does not serve a basic security purpose for the security of the keys [24]. In particular, we again assume that the keys, derived from the handshake secrets are available to the adversary.

The final step is to compute the session key key, given by the client application traffic secret key_{cats} and the server application traffic secret key_{sats} . The additional exporter master secret key_{ems} and resumption master secret key_{rms} are once more irrelevant for us and can be made available to the adversary. The stealth key is now computed by swapping the nonces and the Diffie-Hellman shares, i.e., using nonces $N_C^* \leftarrow g^x$ and $N_S^* \leftarrow g^y$ and key shares $g^a \leftarrow \text{Embd}_{256}^{-1}(N_C)$ and $g^b \leftarrow \text{Embd}_{256}^{-1}(N_S)$ with Diffie-Hellman key g^{ab} . Run the signature steps and the key derivation steps as in the original protocol for these swapped values.

Since we give a reduction for our stealth version to TLS 1.3 directly, we do not detail the multiple key derivation steps in the protocol. Instead, we represent them abstractly as a key derivation function $\text{KDF}(\text{'derive'}, \text{IKM}, \text{context})$, applied in a certain derivation context 'derive' for intermediate keying material IKM (in our setting, a Diffie-Hellman value) and context information, namely the transcript hash over all previously exchanged communication data. In TLS 1.3 this key derivation is implemented via nested executions of the HKDF key derivation function.

For the description of security game it remains to specify the session identifier and the admissible auxiliary input aux. As in [24] the session identifier is given by the communication transcript,

$$\text{sid} = (\text{CH}, \text{CKS}, \text{SH}, \text{SKS}, [\text{SCERT}, \sigma_S], \text{SFIN}, [\text{CCERT}, \sigma_C], \text{CFIN}),$$

containing the authentication data if the parties authenticate.

For the auxiliary information we demand that $\text{aux} = (\text{eph}, \text{sk})$ contains the ephemeral Diffie-Hellman secret $x \in \mathbb{Z}_q$ to be used, as well as the long-term signing key sk of the party if authentication is required and Init is called with party $\neq *$ (else $\text{sk} = \perp$ is admissible). We also require that secret keys are uniquely determined given the public key, and that the correctness of the secret key can be checked efficiently. This holds for example for the ECDSA algorithm or the RSA-PSS algorithm (if the secret key is given in the factorization-based representation), and assuming the collision resistance of the deployed hash function, also for EdDSA. All these algorithms are proposed by TLS 1.3 as admissible signature algorithms [50]. We let the protocol immediately abort if the input aux contains improper values in this regard. If the data are sound then the party can execute the protocol entirely with these given cryptographic values.

4.2 Embedding

We briefly discuss one option for the embedding algorithm Embd here. It closely follows the Elligator 2 approach in [8]. This embedding can be applied for instance to Curve25519 [7] which is one of the elliptic curve options in TLS 1.3 [50]. Other options exist, such as Elligator 1. Abstractly we need that the random mapping Embd maps a large portion of the elliptic curve points to a string which is statistically close to a uniform string. We denote by Δ_{Embd}^n the statistical distance to uniformly distributed n -bit strings.

Curve25519 is the elliptic curve $y^2 = x^3 + Ax^2 + Bx \bmod q$ for $A = 486662$, $B = 1$, and prime $q = 2^{255} - 19$. For this curve Bernstein et al. [8] design an injective mapping $\iota : S \rightarrow E(\mathbb{F}_q)$ from a set S of strings to the elliptic curve. Here the set S can be described by a standard encoding σ of bit strings of length $b = \lfloor \log q \rfloor = 254$ into elements from \mathbb{F}_q , namely, $\sigma(x_0 \dots x_{b-1}) = \sum x_i 2^i$. We assume that each string is encoded with leading 0's to consist of exactly b

bits. Now S is defined as $S = \sigma^{-1}(\{0, 1, \dots, (q-1)/2\})$. Note that by the choice of q these are all bit strings of length 254, except for a negligible subset.

Given S and σ , one can define the embedding $\psi : \mathbb{F}_q \rightarrow E(\mathbb{F}_q)$ as follows. Let u be a non-square in \mathbb{F}_q (like $u = 2$ for Curve25519) and $\sqrt{\cdot}$ be a square-root function over \mathbb{F}_q (e.g., taking the element from 0 to $(q-1)/2$ for the two roots $a, -a$ for some a^2). Let $\chi : \mathbb{F}_q \rightarrow \{\pm 1, 0\}$ defined as $\chi(a) = a^{(q-1)/2}$ indicate if a is zero ($\chi(a) = 0$), a non-zero square ($\chi(a) = 1$), or a non-square ($\chi(a) = -1$). For any $r \in \mathbb{F}_q^*$ set

$$\begin{aligned} v &\leftarrow -A/(1+ur), \quad \epsilon \leftarrow \chi(v^3 + Av^2 + Bv), \\ x &\leftarrow \epsilon v - (1-\epsilon)A/2, \quad y \leftarrow -\epsilon \sqrt{x^3 + Ax^2 + Bx}. \end{aligned}$$

Then $\psi(r) = (x, y)$ describes the curve point for r . One additionally sets $\psi(0) = (0, 0)$ such that ψ is now defined over \mathbb{F}_q .

Set $\iota := \psi \circ \sigma$. For the inverse $\psi^{-1} : \psi(\mathbb{F}_q) \rightarrow \mathbb{F}_q$ define $\sqrt{\cdot}^2_q$ as the set of preimages of squares under $\sqrt{\cdot}$, and

$$\psi^{-1}((x, y)) \leftarrow \begin{cases} \sqrt{-x/((x+A)u)} & y \in \sqrt{\mathbb{F}_q^2} \\ \sqrt{-(x+A)/ux} & y \notin \sqrt{\mathbb{F}_q^2} \end{cases}.$$

This also defines the inverse $\iota^{-1} := \sigma^{-1} \circ \psi^{-1}$. Note that since ι is injective around half of the elliptic curve points have a preimage under ψ . Hence, when picking an elliptic curve point we need on the average two attempts to find a point in the range of ψ .

For our application to stealth TLS we are not entirely done yet. Recall that ψ maps 254-bit strings to elliptic curve points such that, when applying ψ^{-1} to a suitable random curve point P , we get an almost uniform 254-bit string. Our algorithm $\text{Embd}_{256}(P)$ now simply computes $\psi^{-1}(P)$ and appends two random high-order bits. The (deterministic) inverse $\text{Embd}_{256}^{-1}(s)$ drops these two bits and applies ψ to the remaining string.

As pointed out in [8] the sampling via ψ^{-1} and thus via Embd_{256} is statistically close to uniform. This is due to the fact that the order of the field is $2^{255} - 19$ and thus $(q+1)/2$ very close to 2^{254} . Another point is that the actual Curve25519 works in a prime order subgroup (with cofactor 8), such that extra care must be taken to hide public keys in strings if using the genuine Curve25519 algorithms. One option is then to use a base point generating the full group instead, the other option is to add a low-order point to the Curve25519 point. See Loup Valliant's page loupvalliant.org for more implementation details. Let us point out that the deployment of the embedding may introduce timing-based side channels. Since the embedding is computationally more expensive than simply picking nonces, this may reveal if the party runs in stealth mode via time measurements. We neglect this issue here since previous analyses of TLS 1.3 did not consider such side channels or randomness leakage either.

4.3 Advanced Security Features

As explained, our goal is not to re-prove TLS security, but instead to give a reduction from the indistinguishability of our stealth variant to the key secrecy of the regular version of TLS. By construction, the stealth key computation can be thought of as a TLS version in which we swap the nonce and curve point for deriving the key. It is therefore natural to define a swapped version of TLS, denoted swTLS 1.3, which already includes the exchange of the two values

for computing the key. Our security proof will then use a reduction to the regular TLS 1.3 protocol for attacking the session key, and to the swapped version swTLS 1.3 for attacking the stealth key. Both protocols are required to provide key secrecy against adversary which can determine nonces.

Key Indistinguishability against Nonce-Setting Adversaries. The first requirement, for both TLS 1.3 and swTLS 1.3, says that key secrecy still holds if we let the adversary determine the nonce value in executions of the honest parties. This appears to be a reasonable assumption in light of previous results about TLS 1.3. That is, Dowling et al. [24] do not make any assumptions about the nonces in the key secrecy proof (but only for session matching). Davis and Günther [17] only require that the pair of nonce and ephemeral group element is unique in their tight key secrecy proof. If we let the adversary determine the nonce then the minor term for collisions in their security bound decreases from $S^2 \cdot 2^{-256} \cdot \frac{1}{q}$ to $S^2 \cdot \frac{1}{q}$ for the number S of executions. Only the result by Diemert and Jäger [22] in their tightness result about key secrecy uses that the nonces are unique.

Formally, we need to specify how an adversary \mathcal{B} can interact with the standard TLS protocol, and here we mean our stealth TLS protocol in mode regular (with the intermediate keys being immediately exposed). Adversary \mathcal{B} is also allowed to choose nonces. The experiment is almost identical to our model for stealth attacks, with two exceptions:

- Init, Reveal, and Test do not take an additional input mode (since TLS 1.3 only runs in regular mode).
- Init does not take the optional aux input. Instead, it takes an optional nonce input which the session owner then uses as a nonce in the protocol execution. The stipulation here is that \mathcal{B} never chooses the same value nonce twice.

We note that formally we can subsume the changes under our model by always requiring mode = regular for each oracle call and session, and by interpreting the optional aux as the optional nonce input. The latter is admissible because it depends on the protocol what to do with this input, if present. We accordingly write $\mathbf{Exp}_{\mathcal{A}, \Pi, \text{KGen}, \mathcal{U}}^{\text{Secrecy-NS}}$ (NS for nonce setting) for the adversary winning this experiment in predicting the challenge bit b and obeying the other restrictions.

The Swapped TLS Protocol. We next discuss the swTLS 1.3 variant and its security. In this variant we exchange the nonce in the hello messages with the key share value in all subsequent evaluations of the signature algorithm and the key derivation function. In our presentation of the core protocol messages where the hello message only consists of the nonce:

$$\text{CH}|\text{CKS}|\text{SH}|\text{SKS} \mapsto \text{CKS}|\text{CH}|\text{SKS}|\text{SH}$$

in all applications of KDF and of Sign. Again, strictly speaking the key shares are part of the hello messages. According to that terminology we exchange the key share entry with the nonce entry in the hello messages. We leave all other steps unchanged, including also session identifiers.

We note that we do not require TLS 1.3 to be secure in the original and in the swapped order *simultaneously*. Indeed, this infringes with any of the known proofs in [17, 22, 24] which require the input to the signature to be unique, whereas adaptive swapping could

easily violate this. We only require that both TLS 1.3 and swTLS 1.3 are individually secure according to the nonce-setting key secrecy experiment above.

Once more, consulting [17, 24], the security proofs show key secrecy (in the nonce-setting scenario) for swTLS 1.3 as well, assuming the hardness of the underlying Diffie-Hellman problem and security of the deployed cryptographic primitives. The reason is that these proofs rely on abstract collision-resistance of the hash function for the transcript hash used in key derivation and signing. Since (bijectively) changing the order of the inputs does not infringe with collision resistance, these results also show security of the swapped version.

Another property of swTLS 1.3 we require is that we are also able to swap nonce values nonce with elliptic curve points Z in the hello messages. For this we extract the nonce-embedded point $\text{Emb}_n^{-1}(\text{nonce})$ again, and vice versa interpret the point Z as a 256-bit nonce value. The latter is possible by assumption about the deployed group and holds for instance for Curve25519. This swapping has the effect that we now work with a Diffie-Hellman problem over “embeddable” points only. Nevertheless, it is reasonable to assume for Curve25519 and Elligator 2 that the problem is still hard, since half of the points allow for such an embedding.

5 SECURITY PROOF OF STEALTH TLS 1.3

We present here the security statements for our stealth protocol. We note that correctness of sTeaLS holds obviously. If two parties faithfully execute the protocol, then they obtain the same session identifier. With the session matching property below it follows that they also have the same session and stealth keys then.

The first property is session matching. The proof follows the approaches for the original TLS 1.3 protocol closely and can be found in the full version [26].

PROPOSITION 5.1. *Let sTeaLS be the stealth TLS 1.3 protocol (for a set of users \mathcal{U} and key generation algorithm KGen). Then for any adversary \mathcal{A} initializing at most S sessions we have*

$$\Pr \left[\mathbf{Game}_{\mathcal{A}, \text{sTeaLS}, \text{KGen}, \mathcal{U}}^{\text{Match}} \right] \leq S^2 \cdot \frac{1}{q} \cdot 2^{-n} + S \cdot \Delta_{\text{Emb}}^n,$$

where $n = 256$ is the nonce length, q is the size of the underlying elliptic curve, and Δ_{Emb}^n is the statistical distance from uniform for the embedding algorithm in sTeaLS.

The quadratic term originates from the collision bound, and the term $S \cdot \Delta_{\text{Emb}}^n$ from the fact that in stealth mode we are not using uniformly distributed nonces but embedded values instead.

The indistinguishability proof is more elaborate. Recall that we reduce the security of the stealth protocol to the security of TLS 1.3 resp. swTLS 1.3 in the nonce-setting scenario. For the theorem’s statement it is convenient to denote by

$$\mathbf{Adv}_{\mathcal{A}, \Pi, \text{KGen}, \mathcal{U}}^x := \Pr \left[\mathbf{Exp}_{\mathcal{A}, \Pi, \text{KGen}, \mathcal{U}}^x = 1 \right] - \frac{1}{2}$$

the advantage over the guessing probability for any type of experiment.

THEOREM 5.2. *For any key generation KGen algorithm and user set \mathcal{U} , and any adversary \mathcal{A} initializing at most S sessions, there*

exist adversaries \mathcal{B} and \mathcal{C} (with roughly the same efficiency as \mathcal{A}) such that

$$\text{Adv}_{\mathcal{A}, s\text{TeaLS}, \text{KGen}, \mathcal{U}}^{\text{Ind}} \leq 2S \cdot \left(\text{Adv}_{\mathcal{B}, \text{TLS 1.3}, \text{KGen}, \mathcal{U}}^{\text{Secrecy-NS}} + \text{Adv}_{\mathcal{C}, \text{swTLS 1.3}, \text{KGen}, \mathcal{U}}^{\text{Secrecy-NS}} \right) + S \cdot \Delta_{\text{Emb}}^n$$

where $n = 256$, q is the order of the group, and Δ_{Emb}^n is the statistical distance from uniform for the embedding algorithm in $s\text{TeaLS}$.

Once more, the proof can be found in the full version [26]. The idea is to restrict the adversary to a single Test query and guess the tested session and its type, regular or stealth, in advance. This can be done with a standard reduction, introducing a factor $2S$ in the security bound. Then we either give a reduction \mathcal{B} against the original TLS protocol (if the tested session is of type regular) or a reduction \mathcal{C} against the swapped version (if the type is stealth). The additional term $S \cdot \Delta_{\text{Emb}}^n$ comes from the statistical distance of the embedding.

6 INTEGRATION INTO THE TLS 1.3 RECORD PROTOCOL

In this section we describe how one can use the stealth key exchange to derive a sanitizable channel. The full description of the construction of the sanitizable channel and the security proofs can be found in the full version [26]. In this overview we only describe a sanitizable version of the TLS 1.3 record layer in which the sanitizer has partly access to designated parts of the record protocol data.

Key Establishment. We assume that the sender and the receiver have executed the TLS 1.3 key exchange protocol. The two parties have used the stealth mode to generate a stealth key stkey in addition to the session key chkey . This is done in such a way that the sanitizer also knows this key chkey (but the sanitizer remains oblivious about the stealth key). One option was to let the receiver securely pass the session key to the sanitizer upon establishment, albeit this appears to be very inconvenient in the firewall setting. An alternative is to let the sanitizer provide the ephemeral secret of the receiver in the key exchange step, being able to compute chkey from the transcript of communication. This requires the sanitizer to either communicate with the receiver while the key exchange protocol runs, or by sharing a local key with the receiver from which the ephemeral secret is derived. Alternatively, the receiver may re-use a sanitizer-provided ephemeral secret in multiple executions. In fact, this corresponds to the static Diffie-Hellman share solution for TLS 1.3 [30]. The disadvantage in the latter case is that this solution infringes with forward secrecy (yet, forward secrecy in the stealth part of the connection is still preserved).

TLS 1.3 Record Protocol. We note that the key in TLS 1.3 consists of the actual encryption and decryption key and a random offset, called `client_write_iv` resp. `server_write_iv` in TLS, depending on the direction of communication. In this sense it is understood that our derived keys chkey and stkey both contain such a random offset.

The key and the offset are used to encrypt the payload message m in the TLS 1.3 record protocol via a scheme for authenticated encryption with associated data (AEAD) [51] as $c \leftarrow \text{AEEnc}(\text{key}, \text{offset} \oplus \text{st}_S, \text{AD}, m)$. Here, $\text{offset} \oplus \text{st}_S$ is used as a nonce for the AEAD

scheme and st_S is a counter (the state of the sender), incremented with each sent ciphertext. The associated data in TLS 1.3 are given by the constant `ContentType opaque_type = application_data` (which equals 23), followed by the constant `ProtocolVersion legacy_record_version = 0x0303`, followed by the (expected) length of the ciphertext in bytes. The latter can be derived from the length of the input message m for the suggested AEAD schemes. To decrypt the receiver calls $m \leftarrow \text{AEDec}(\text{key}, \text{offset} \oplus \text{st}_R, \text{AD}, m)$ where st_R is the current counter value of the receiver (incremented, too, after successful decryption) and AD is given by the constants and ciphertext length as for encryption.

Partly Accessible Channel. We can now proceed as follows to build the stealth channel. Recall that sender, receiver, and sanitizer all share the session key chkey (including the offset choffset), but only sender and receiver know the stealth key stkey (with its own offset stoffset). We assume that we have a message part m_{sec} which should be sent confidentially between sender and receiver, and a part m_{plain} which should only be accessible by the sanitizer (but not to outsiders). We now use a nested encryption, encrypting the m_{sec} -part under the stealth key and then the derived ciphertext together with m_{plain} under the channel key:

$$\begin{aligned} c_{\text{sec}} &\leftarrow \text{AEEnc}(\text{stkey}, \text{stoffset} \oplus \text{st}_S, \text{AD}, m_{\text{sec}}) \\ c &\leftarrow \text{AEEnc}(\text{chkey}, \text{choffset} \oplus \text{st}_S, \text{AD}', (c_{\text{sec}}, m_{\text{plain}})) \end{aligned}$$

Here AD and AD' are the corresponding associated data.

We note that, with this construction, the sanitizer may alter the m_{plain} -part. If we want to give read-only access to the message part, then we put m_{plain} into the associated data ($\text{AD}, m_{\text{plain}}$) in the inner encryption. Since the associated data are authenticated via the stealth key stkey , the sanitizer cannot modify m_{plain} without the receiver detecting modifications of m_{plain} . In fact, this means we rather put the accessible part in m_{auth} and leave m_{plain} empty, according to the terminology of message parts in sanitizable channels as described in the full version [26].

The sender then transmits c . The receiver and the sanitizer can individually recover $(c_{\text{sec}}, m_{\text{plain}})$ with the help of chkey . The sanitizer can check the information in m_{plain} , but only the receiver is able to also recover the message m_{sec} from c_{sec} with the help of stkey . If m_{plain} is part of the associated data in c_{sec} , then the receiver can also check its integrity. We note that outsiders, which do not know chkey , cannot access either of the two parts.

There are two things to consider regarding the length of the nested ciphertext c . First note that, compared to subliminal communication, an outsider can observe that ciphertexts in this version are longer than when using the original record protocol. As explained in the introduction, we do not aim to hide this fact. Secondly, TLS 1.3 sets an upper bound of $2^{14} + 256$ bytes for the length of ciphertexts, requiring that input messages are of at most 2^{14} bytes (or else need to be fragmented) [50]. This needs to be taken into account with the ciphertext expansion due to the double encryption here. Indeed, we need to make sure that the combined length of $(c_{\text{sec}}, m_{\text{plain}})$ is at most 2^{14} bytes, resulting in an overall bound of $2^{14} - 256$ for m_{sec} and m_{plain} and possibly further fragmentations. Let us stress once more that our goal is not to hide the fact that we are using the stealth channel. If this is obeyed, then c is a perfectly legitimate

TLS 1.3 record protocol ciphertext which supports read-only access for the sanitizer.

7 TOWARDS INTEGRATION INTO INTRUSION DETECTION SYSTEMS

In this section we describe how one can use our sanitizable channels in combination with a network intrusion detection and prevention system like the well-known open-source system Snort (<https://www.snort.org/>). We assume that the keys have already been established, as described in Section 6 about setting up the sanitizable channel. The reader may for now think of the intrusion detection system using a static Diffie-Hellman key which the receiver obtains when logging into the local network, and which is then used in the stealth key exchange step to establish the channel key (accessible also by the intrusion detection system). The stealth key is only available to the sender and receiver.

Snort in version 3 comes with a set of 4,031 predefined rules, called the Community Ruleset. This set is updated frequently, we refer here to the one of February 6th, 2023. The rules allow to detect malicious network behavior of various types. As an example, consider the rule with identifier *sid 26261* for detecting potential phishing attacks (parts omitted for readability):

```
alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any ( msg: ←
"MALWARE-OTHER Fake postal receipt HTTP Response phishing attack"; ←
flow:to_client,established; http_header; content: ←
"|3B 20|filename=Postal-Receipt.zip|0D 0A|",fast_pattern,nocase; ←
... classtype:trojan-activity; sid:26261; rev:3; )
```

The rule checks if the incoming network traffic on HTTP ports contains suspicious file names in the HTTP header. The HTTP header contains meta-information about the actual HTTP content and the sending party. In secured HTTPS connections the header is also encrypted and thus inaccessible to an intrusion detection system like Snort.

With our sanitizable channel protocol, combined with the stealth key exchange, we could give Snort as the sanitizer access to the HTTP header information (and similar meta-data such as the HTTP status code and URI) by placing this information into the m_{plain} -part, protected under the channel key ch_{key} shared also with the sanitizer. We put the HTTP content into the inner m_{sec} -part, protected by the outer channel key ch_{key} as well as the inner stealth key st_{key} only known by the sender and receiver (see Figure 2). Then Snort can access the header information and apply qualified rules, whereas the actual HTTP content remains hidden from Snort. From the outside, the communication still appears to be a valid HTTPS resp. TLS connection, integrating smoothly into existing network environments.

To estimate the usefulness we note that the Community Ruleset currently lists roughly half of the rules with reference to HTTP fields $http_*$ (altogether 2,011 rules). Of this set, 470 rules use the $http_header$ field and *no* reference to the body $http_client_body$. If we also grant Snort access to other HTTP data such as the URI in outgoing traffic via the $http_uri$ flag, or the $http_cookie$ flag for Cookie header information, then the coverage increases significantly. Among the Community Ruleset, 1,776 rules include one

of the $http_*$ fields without listing $http_client_body$. These are 44% of all rules and 88% of all HTTP-related rules.

The solution still comes with some inconveniences, though. First of all, one carefully needs to evaluate if revealing the HTTP information to Snort is admissible. Secondly, scanning the content is still not possible. Third, HTTPS currently does not differentiate between confidentiality levels for the HTTP parts and one would thus need to change the protocol in order to accommodate the specification of different confidentiality levels for data.

8 CONCLUSION

Our results show that, with some extra effort, existing cryptographic mechanisms can be enhanced to enable further features. As for the overhead, we note that we did some initial experiments for the stealth key exchange on commodity hardware. The computational costs in our experiments went up by roughly a factor 2.5 compared to the plain TLS 1.3 handshake protocol. This matches the expected overhead from theory, since one runs roughly two TLS key exchanges, plus Elligator needs two attempts to find a suitable point on the average, plus inversion time for the embedding. Based on the results in [45] about using post-quantum primitives in TLS connections for various network settings, it is plausible that the common network latency will also level out the slowdown due to our stealth computations.

We stress again that the changes to achieve stealthiness in TLS 1.3 require protocol modifications at the end points but not on the network layer. That is, the protocol is fully compatible with common TLS 1.3 network traffic. Still, integrating the sanitizable channel to enable HTTPS scanning as explained in Section 7 asks for modifications on the application-channel interface, for both the HTTPS part—semantically labeling header and body data—as well as on the TLS channel side, processing the different inputs parts accordingly. This certainly poses further engineering challenges. It is, however, beyond our cryptographic treatment here, showing that a graceful access, being fully under control of the sending party, is cryptographically possible.

An interesting prospect in light of stealth channels is the planned deployment of TLS hybrid key exchange protocols, as discussed for example by the IETF [54]. In such a hybrid solution one runs a classical key exchange protocol, e.g., based on Diffie-Hellman, together with a quantum-resistant one, e.g., based on Kyber as suggested in [54]. In this case the design would already generate two keys, and for the common cases one could now implement a stealth version within the given system. For this one either uses the Diffie-Hellman part, either generating a known secret key x by sending g^x , or refraining from doing so by sending $\text{Embd}^{-1}(\text{nonce})$ instead. Alternatively, one could also use the post-quantum part for the same purpose. The latter is possible since Kyber provides strong pseudorandomness under chosen-ciphertext attacks (SPR-CCA) [41], meaning that outsiders cannot distinguish actual ciphertexts from random strings.³ In other words, a sender could deny to be able to compute the shared key in the classical or the quantum-secure part of the protocol. However, as opposed to our solution here which *preserves* security of the original protocol, the sketched

³Another interesting feature in the Kyber case is that the receiver may actually become aware of the sender's choice by trying to decrypt.

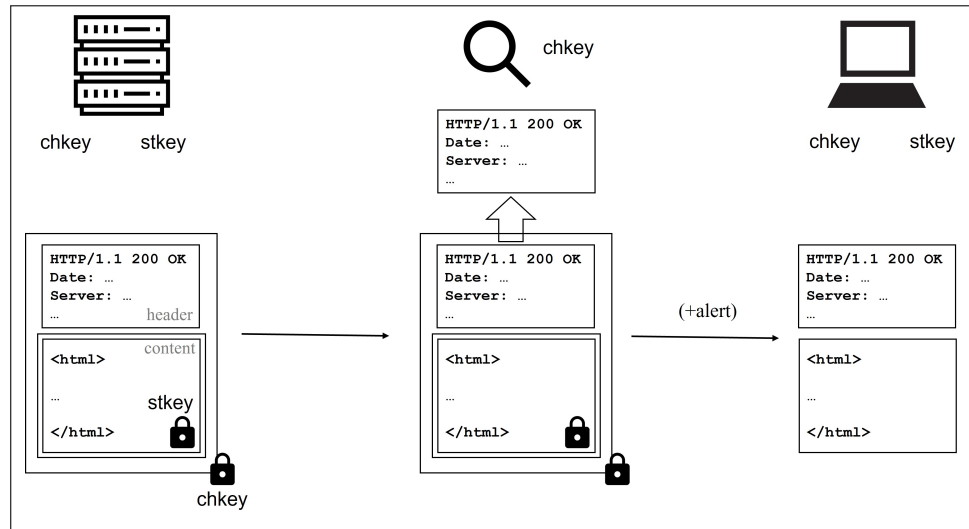


Figure 2: IDS checking HTTP header information in sanitizable channel.

hybrid solution would actually degrade security, although starting from a higher level. That is, the resulting scheme would only be classically secure or be post-quantum secure when using only one key, but would fail to give fallback security—which is the original idea of using hybrid schemes.

ACKNOWLEDGMENTS

We thank the anonymous reviewers and Felix Günther for valuable comments. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – SFB 1119 – 236615297 and by the German Federal Ministry of Education and Research and the Hessian Ministry of Higher Education, Research, Science and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

REFERENCES

- [1] Ross J. Anderson and Fabien A. P. Petitcolas. 1998. On the limits of steganography. *IEEE J. Sel. Areas Commun.* 16, 4 (1998), 474–481. <https://doi.org/10.1109/49.668971>
- [2] Diego F. Aranha, Pierre-Alain Fouque, Chen Qian, Mehdi Tibouchi, and Jean-Christophe Zapolowicz. 2014. Binary Elligator Squared. In *SAC 2014 (LNCS, Vol. 8781)*, Antoine Joux and Amr M. Youssef (Eds.). Springer, Heidelberg, 20–37. https://doi.org/10.1007/978-3-319-13051-4_2
- [3] Giuseppe Ateniese, Daniel H. Chou, Breno de Medeiros, and Gene Tsudik. 2005. Sanitizable Signatures. In *ESORICS 2005 (LNCS, Vol. 3679)*, Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann (Eds.). Springer, Heidelberg, 159–177. https://doi.org/10.1007/11555827_10
- [4] Michael Backes and Christian Cachin. 2005. Public-Key Steganography with Active Attacks. In *TCC 2005 (LNCS, Vol. 3378)*, Joe Kilian (Ed.). Springer, Heidelberg, 210–226. https://doi.org/10.1007/978-3-540-30576-7_12
- [5] Mihir Bellare and Phillip Rogaway. 1994. Entity Authentication and Key Distribution. In *CRYPTO'93 (LNCS, Vol. 773)*, Douglas R. Stinson (Ed.). Springer, Heidelberg, 232–249. https://doi.org/10.1007/3-540-48329-2_21
- [6] Sebastian Berndt and Maciej Liskiewicz. 2018. On the Gold Standard for Security of Universal Steganography. In *EUROCRYPT 2018, Part I (LNCS, Vol. 10820)*, Jesper Buus Nielsen and Vincent Rijmen (Eds.). Springer, Heidelberg, 29–60. https://doi.org/10.1007/978-3-319-78381-9_2
- [7] Daniel J. Bernstein. 2006. Curve25519: New Diffie-Hellman Speed Records. In *PKC 2006 (LNCS, Vol. 3958)*, Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin (Eds.). Springer, Heidelberg, 207–228. https://doi.org/10.1007/11745853_14
- [8] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. 2013. Elligator: elliptic-curve points indistinguishable from uniform random strings. In *ACM CCS 2013*, Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung (Eds.). ACM Press, 967–980. <https://doi.org/10.1145/2508859.2516734>
- [9] Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. 2015. A Messy State of the Union: Taming the Composite State Machines of TLS. In *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 535–552. <https://doi.org/10.1109/SP.2015.39>
- [10] Karthikeyan Bhargavan, Cédric Fournet, and Markulf Kohlweiss. 2016. miTLS: Verifying Protocol Implementations against Real-World Attacks. *IEEE Secur. Priv.* 14, 6 (2016), 18–25. <https://doi.org/10.1109/MSP.2016.123>
- [11] David Cerdeira, Nuno Santos, Pedro Fonseca, and Sandro Pinto. 2020. SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems. In *2020 IEEE Symposium on Security and Privacy, SP 2020*. IEEE, 1416–1432. <https://doi.org/10.1109/SP40000.2020.00061>
- [12] Scott Craver. 1998. On Public-Key Steganography in the Presence of an Active Warden. In *Information Hiding, Second International Workshop, Portland, Oregon, USA, April 14–17, 1998, Proceedings (Lecture Notes in Computer Science, Vol. 1525)*, David Aucsmith (Ed.). Springer, 355–368. https://doi.org/10.1007/3-540-49380-8_25
- [13] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. 2017. A Comprehensive Symbolic Analysis of TLS 1.3. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 1773–1788. <https://doi.org/10.1145/3133956.3134063>
- [14] Cas Cremers, Marko Horvat, Sam Scott, and Thyla van der Merwe. 2016. Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication. In *2016 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 470–485. <https://doi.org/10.1109/SP.2016.35>
- [15] Ivan Damgård, Helene Haagh, and Claudio Orlandi. 2016. Access Control Encryption: Enforcing Information Flow with Cryptography. In *TCC 2016-B, Part II (LNCS, Vol. 9986)*, Martin Hirt and Adam D. Smith (Eds.). Springer, Heidelberg, 547–576. https://doi.org/10.1007/978-3-662-53644-5_21
- [16] Hannah Davis, Denis Diemert, Felix Günther, and Tibor Jäger. 2022. On the Concrete Security of TLS 1.3 PSK Mode. In *EUROCRYPT 2022, Part II (LNCS, Vol. 13276)*, Orr Dunkelman and Stefan Dziembowski (Eds.). Springer, Heidelberg, 876–906. https://doi.org/10.1007/978-3-031-07085-3_30
- [17] Hannah Davis and Felix Günther. 2021. Tighter Proofs for the SIGMA and TLS 1.3 Key Exchange Protocols. In *Applied Cryptography and Network Security (ACNS), 2021 (Lecture Notes in Computer Science, Vol. 12727)*, Kazuo Sako and Nils Ole Tippenhauer (Eds.). Springer, 448–479. https://doi.org/10.1007/978-3-030-78375-4_18
- [18] Xavier de Carné de Carnavalet and Mohammad Mannan. 2016. Killed by Proxy: Analyzing Client-end TLS Interce. In *23rd Annual Network and Distributed System Security Symposium, NDSS*. The Internet Society. <http://www.internetsociety.org/events/ndss-symposium-2016>
- [19] Xavier de Carné de Carnavalet and Paul C. van Oorschot. 2020. A survey and analysis of TLS interception mechanisms and motivations. *CoRR abs/2010.16388* (2020). arXiv:2010.16388 <https://arxiv.org/abs/2010.16388>

- [20] Nenad Dedic, Gene Itkis, Leonid Reyzin, and Scott Russell. 2009. Upper and Lower Bounds on Black-Box Steganography. *Journal of Cryptology* 22, 3 (July 2009), 365–394. <https://doi.org/10.1007/s00145-008-9020-3>
- [21] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Jonathan Protzenko, Aseem Rastogi, Nikhil Swamy, Santiago Zanella-Béguelin, Karthikeyan Bhargavan, Jianyang Pan, and Jean Karim Zinzindohoue. 2017. Implementing and Proving the TLS 1.3 Record Layer. In *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 463–482. <https://doi.org/10.1109/SP.2017.58>
- [22] Denis Diemert and Tibor Jager. 2021. On the Tight Security of TLS 1.3: Theoretically Sound Cryptographic Parameters for Real-World Deployments. *Journal of Cryptology* 34, 3 (July 2021), 30. <https://doi.org/10.1007/s00145-021-09388-x>
- [23] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. 2015. A Cryptographic Analysis of the TLS 1.3 Handshake Protocol Candidates. In *ACM CCS 2015*, Indrajit Ray, Ninghui Li, and Christopher Kruegel (Eds.). ACM Press, 1197–1210. <https://doi.org/10.1145/2810103.2813653>
- [24] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. 2021. A Cryptographic Analysis of the TLS 1.3 Handshake Protocol. *Journal of Cryptology* 34, 4 (Oct. 2021), 37. <https://doi.org/10.1007/s00145-021-09384-1>
- [25] Victoria Fehr and Marc Fischlin. 2015. Sanitizable Signcryption: Sanitization over Encrypted Data (Full Version). Cryptology ePrint Archive, Report 2015/765. <https://eprint.iacr.org/2015/765>.
- [26] Marc Fischlin. 2023. Stealth Key Exchange and Confined Access to the Record Protocol Data in TLS 1.3. Cryptology ePrint Archive, Paper 2023/651. <https://eprint.iacr.org/2023/651>
- [27] Marc Fischlin and Felix Günther. 2014. Multi-Stage Key Exchange and the Case of Google's QUIC Protocol. In *ACM CCS 2014*, Gail-Joon Ahn, Moti Yung, and Ninghui Li (Eds.). ACM Press, 1193–1204. <https://doi.org/10.1145/2660267.2660308>
- [28] Pierre-Alain Fouque, Antoine Joux, and Mehdi Tibouchi. 2013. Injective Encodings to Elliptic Curves. In *ACISP 13 (LNCS, Vol. 7959)*, Colin Boyd and Leonie Simpson (Eds.). Springer, Heidelberg, 203–218. https://doi.org/10.1007/978-3-642-39059-3_14
- [29] Georg Fuchsbauer, Romain Gay, Lucas Kowalczyk, and Claudio Orlandi. 2017. Access Control Encryption for Equality, Comparison, and More. In *PKC 2017, Part II (LNCS, Vol. 10175)*, Serge Fehr (Ed.). Springer, Heidelberg, 88–118. https://doi.org/10.1007/978-3-662-54388-7_4
- [30] Matthew Green, Ralph Droms, Russ Housley, Paul Turner, and Steve Fenter. 2017. Data Center use of Static Diffie-Hellman in TLS 1.3. Internet-Draft draft-green-tls-static-dh-in-tls13-01. Internet Engineering Task Force. <https://datatracker.ietf.org/doc/html/draft-green-tls-static-dh-in-tls13-01> Work in Progress.
- [31] Paul Grubbs, Arasu Arun, Ye Zhang, Joseph Bonneau, and Michael Walfish. 2021. Zero-Knowledge Middleboxes. *IACR Cryptol. ePrint Arch.* (2021), 1022. <https://eprint.iacr.org/2021/1022>
- [32] Nicholas Hopper. 2005. On Steganographic Chosen Covert Security. In *ICALP 2005 (LNCS, Vol. 3580)*, Luis Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung (Eds.). Springer, Heidelberg, 311–323. https://doi.org/10.1007/11523468_26
- [33] Amir Houmansadr, Giang T. K. Nguyen, Matthew Caesar, and Nikita Borisov. 2011. Cirripede: circumvention infrastructure using router redirection with plausible deniability. In *ACM CCS 2011*, Yan Chen, George Danezis, and Vitaly Shmatikov (Eds.). ACM Press, 187–200. <https://doi.org/10.1145/2046707.2046730>
- [34] Josh Karlin, Daniel Ellard, Alden W. Jackson, Christine E. Jones, Greg Lauer, David Mankins, and W. Timothy Strayer. 2011. Decoy Routing: Toward Unblockable Internet Communication. In *USENIX Workshop on Free and Open Communications on the Internet, FOCI '11, San Francisco, CA, USA, August 8, 2011*, Nick Feamster and Wenke Lee (Eds.). USENIX Association. <https://www.usenix.org/conference/foci11/decoy-routing-toward-unblockable-internet-communication>
- [35] Sam Kim and David J. Wu. 2017. Access Control Encryption for General Policies from Standard Assumptions. In *ASIACRYPT 2017, Part I (LNCS, Vol. 10624)*, Tsuyoshi Takagi and Thomas Peyrin (Eds.). Springer, Heidelberg, 471–501. https://doi.org/10.1007/978-3-319-70694-8_17
- [36] Markulf Kohlweiss, Ueli Maurer, Cristina Onete, Björn Tackmann, and Daniele Venturi. 2015. (De-)Constructing TLS 1.3. In *INDOCRYPT 2015 (LNCS, Vol. 9462)*, Alex Biryukov and Vipul Goyal (Eds.). Springer, Heidelberg, 85–102. https://doi.org/10.1007/978-3-319-26617-6_5
- [37] Hugo Krawczyk and Hoeteck Wee. 2016. The OPTLS Protocol and TLS 1.3. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*. IEEE, 81–96. <https://doi.org/10.1109/EuroSP.2016.18>
- [38] Mirosław Kutylowski, Giuseppe Persiano, Duong Hieu Phan, Moti Yung, and Marcin Zawada. 2023. Anamorphic Signatures: Secrecy from a Dictator Who Only Permits Authentication! , 759–790 pages.
- [39] Tri Van Le and Kaoru Kurosawa. 2006. Bandwidth Optimal Steganography Secure Against Adaptive Chosen Stegotext Attacks. In *Information Hiding, 8th International Workshop, IH 2006, Alexandria, VA, USA, July 10-12, 2006. Revised Selected Papers (Lecture Notes in Computer Science, Vol. 4437)*, Jan Camenisch, Christian S. Collberg, Neil F. Johnson, and Phil Sallee (Eds.). Springer, 297–313. https://doi.org/10.1007/978-3-540-74124-4_20
- [40] Hyunwoo Lee, Zach Smith, Junghwan Lim, Gyeongjae Choi, Selin Chun, Taejoong Chung, and Ted Taekyoung Kwon. 2019. maTLS: How to Make TLS middlebox-aware?. In *NDSS*. The Internet Society.
- [41] Varun Maram and Keita Xagawa. 2023. Post-quantum Anonymity of Kyber. In *PKC 2023, Part I (LNCS, Vol. 13940)*, Alexandra Boldyreva and Vladimir Kolesnikov (Eds.). Springer, Heidelberg, 3–35. https://doi.org/10.1007/978-3-031-31368-4_1
- [42] David A. McGrew and John Viega. 2004. The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In *INDOCRYPT 2004 (LNCS, Vol. 3348)*, Anne Canteaut and Kapalee Viswanathan (Eds.). Springer, Heidelberg, 343–355.
- [43] Bodo Möller. 2004. A Public-Key Encryption Scheme with Pseudo-random Ciphertexts. In *ESORICS 2004 (LNCS, Vol. 3193)*, Pierangela Samarati, Peter Y. A. Ryan, Dieter Gollmann, and Refik Molva (Eds.). Springer, Heidelberg, 335–351. https://doi.org/10.1007/978-3-540-30108-0_21
- [44] David Naylor, Kyle Schomp, Matteo Varvello, Ilias Leontiadis, Jeremy Blackburn, Diego R. López, Konstantina Papagiannaki, Pablo Rodriguez Rodriguez, and Peter Steenkiste. 2015. Multi-Context TLS (mcTLS): Enabling Secure In-Network Functionality in TLS. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2015*. ACM, 199–212. <https://doi.org/10.1145/2785956.2787482>
- [45] Christian Paquin, Douglas Stebila, and Goutam Tamvada. 2020. Benchmarking Post-quantum Cryptography in TLS. In *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020, Jintai Ding and Jean-Pierre Tillich (Eds.)*. Springer, Heidelberg, 72–91. https://doi.org/10.1007/978-3-030-44223-1_5
- [46] Ripon Patgiri and Naresh Babu Muppalaeni. 2022. Stealth: A Highly Secured End-to-End Symmetric Communication Protocol. In *International Symposium on Networks, Computers and Communications, ISNCC 2022, Shenzhen, China, July 19-22, 2022*. IEEE, 1–8. <https://doi.org/10.1109/ISNCC55209.2022.9851810>
- [47] Giuseppe Persiano, Duong Hieu Phan, and Moti Yung. 2022. Anamorphic Encryption: Private Communication Against a Dictator. In *EUROCRYPT 2022, Part II (LNCS, Vol. 13276)*, Orr Dunkelman and Stefan Dziembowski (Eds.). Springer, Heidelberg, 34–63. https://doi.org/10.1007/978-3-031-07085-3_2
- [48] Gordon Procter. 2014. A Security Analysis of the Composition of ChaCha20 and Poly1305. Cryptology ePrint Archive, Report 2014/613. <https://eprint.iacr.org/2014/613>.
- [49] Khan Farhan Rafat. 2019. A Stealth Key Exchange Protocol. 675–695. https://doi.org/10.1007/978-3-030-22868-2_48
- [50] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. <https://doi.org/10.17487/RFC8446>
- [51] Phillip Rogaway. 2002. Authenticated-Encryption With Associated-Data. In *ACM CCS 2002*, Vijayalakshmi Atluri (Ed.). ACM Press, 98–107. <https://doi.org/10.1145/586110.586125>
- [52] Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. 2015. Blind-Box: Deep Packet Inspection over Encrypted Traffic. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2015*. ACM, 213–226. <https://doi.org/10.1145/2785956.2787502>
- [53] Gustavus J. Simmons. 1983. The Prisoners' Problem and the Subliminal Channel. In *CRYPTO '83*, David Chaum (Ed.). Plenum Press, New York, USA, 51–67.
- [54] Douglas Stebila, Scott Fluhrer, and Shay Gueron. 2023. Hybrid key exchange in TLS 1.3. Internet-Draft draft-ietf-tls-hybrid-design-08. Internet Engineering Task Force. <https://datatracker.ietf.org/doc/draft-ietf-tls-hybrid-design/08/> Work in Progress.
- [55] Mehdi Tibouchi. 2014. Elligator Squared: Uniform Points on Elliptic Curves of Prime Order as Uniform Random Strings. In *FC 2014 (LNCS, Vol. 8437)*, Nicolas Christin and Reihaneh Safavi-Naini (Eds.). Springer, Heidelberg, 139–156. https://doi.org/10.1007/978-3-662-45472-5_10
- [56] Luis von Ahn and Nicholas J. Hopper. 2004. Public-Key Steganography. In *EUROCRYPT 2004 (LNCS, Vol. 3027)*, Christian Cachin and Jan Camenisch (Eds.). Springer, Heidelberg, 323–341. https://doi.org/10.1007/978-3-540-24676-3_20
- [57] Xiuhua Wang and Sherman S. M. Chow. 2021. Cross-Domain Access Control Encryption: Arbitrary-policy, Constant-size, Efficient. In *2021 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 748–761. <https://doi.org/10.1109/SP40001.2021.00023>
- [58] Zachary Weinberg, Jeffrey Wang, Vinod Yegneswaran, Linda Briesemeister, Steven Cheung, Frank Wang, and Dan Boneh. 2012. StegoTorus: a camouflage proxy for the Tor anonymity system. In *ACM CCS 2012*, Ting Yu, George Danezis, and Virgil D. Gligor (Eds.). ACM Press, 109–120. <https://doi.org/10.1145/2382196.2382211>
- [59] Eric Wustrow, Colleen Swanson, and J. Alex Halderman. 2014. TapDance: End-to-Middle Anticensorship without Flow Blocking. In *USENIX Security 2014*, Kevin Fu and Jaeyeon Jung (Eds.). USENIX Association, 159–174.
- [60] Eric Wustrow, Scott Wolchok, Ian Goldberg, and J. Alex Halderman. 2011. Telex: Anticensorship in the Network Infrastructure. In *USENIX Security 2011*. USENIX Association.