Hidden Stream Ciphers and TMTO Attacks on TLS 1.3, DTLS 1.3, QUIC, and Signal

John Preuß Mattsson

Ericsson Research, Stockholm, Sweden
john.mattsson@ericsson.com

Abstract. Transport Layer Security (TLS) 1.3 and the Signal protocol are very important and widely used security protocols. We show that the key update function in TLS 1.3 and the symmetric key ratchet in Signal can be modeled as non-additive synchronous stream ciphers. This means that the efficient Time Memory Tradeoff Attacks for stream ciphers can be applied. The implication is that TLS 1.3, QUIC, DTLS 1.3, and Signal offer a lower security level against TMTO attacks than expected from the key sizes. We provide detailed analyses of the key update mechanisms in TLS 1.3 and Signal, illustrate the importance of ephemeral key exchange, and show that the process that DTLS 1.3 and QUIC use to calculate AEAD limits is flawed. We provide many concrete recommendations for the analyzed protocols.

Keywords: TLS 1.3 · QUIC · DTLS 1.3 · Signal · Secret-key Cryptography · Key Derivation · Ratchet · Key Chain · Stream Cipher · Key Space · TMTO

1 Introduction

Transport Layer Security (TLS) is the single most important security protocol in the information and communications technology industry. The latest version, TLS 1.3 [26] is already widely deployed and is the default version on the Web and in many other industries. Several other very important protocols such as QUIC [16], EAP-TLS 1.3 [25], DTLS 1.3 [28], DTLS-SRTP [20], and DTLS/SCTP [32] are based on the TLS 1.3 handshake. The US National Institute of Standards and Technology (NIST) requires support for TLS 1.3 by January 1, 2024 [21]. Two nodes that support TLS 1.3 will never negotiate the obsolete TLS 1.2.

The Signal protocol [31] is very popular for end-to-end encryption of voice calls and instant messaging conversations. In addition to the Signal messaging service itself, the Signal protocol is used in WhatsApp, Meta Messenger, and Android Messages. The Signal messaging service is approved for use by the U.S. Senate and is recommended for the staff at the European Commission.

For efficient forward secure symmetric rekeying without Diffie-Hellman, TLS 1.3 and the Signal protocol use symmetric key ratchets in which a deterministic Key Derivation Function (KDF) $H()$ is frequently used to update and replace the current key $k = H(k)$. In Sect. 3 we show that the key update function

in TLS 1.3 and the symmetric key ratchet in Signal can be modeled as non-additive synchronous stream ciphers. As the state sizes in the TLS 1.3 and Signal ratchets are often less than twice the expected security level, the efficient Time Memory Tradeoff Attacks for stream ciphers can be applied [3, 8]. The implication is that TLS 1.3, QUIC, DTLS 1.3, and Signal offer a lower security level against TMTO attacks than expected from the key sizes. In Sects. 4 and 5 we provide detailed analyses of the key update mechanisms in TLS 1.3 and Signal, illustrate the importance of ephemeral key exchange, show that the traffic secrets in TLS_CHACHA20_POLY1305_SHA256 are so small that the the nonce randomization does not improve multi-key security, show that early messages in Signal may not provide forward secrecy with respect to long-term keys, and show that the process that DTLS 1.3 and QUIC use to calculate AEAD limits is flawed. We provide many concrete recommendations for the analyzed protocols. The upcoming revisions of the TLS 1.3 protocol [27] and DTLS/SCTP [33] have already been updated based on this work, see Sect. 5.2.

2 Preliminaries

2.1 Signal Protocol and the Symmetric-Key Ratchet

The Signal protocol [7, 9, 31] consists of the Extended Triple Diffie-Hellman (X3DH) key agreement protocol and the Double Ratchet algorithm. The Double Ratchet algorithm consists of a symmetric-key ratchet and a Diffie-Hellman ratchet. After the X3DH handshake is finished and at least one step of the Double Ratchet has been performed, a 256-bit initial chain key k_0 is derived (to simplify things we only discuss one of the directions). A chain of keys $k_0, k_1, k_2 \ldots$ derived from the initial chain key k_0 are used to protect all future messages sent in one direction until the Diffie-Hellman ratchet is used again. Message i is encrypted using a 256-bit message key K_i and an AEAD algorithm without nonce. Each message key K_i is only used once. The associated data contains identity information for both parties.

Before each message is sent, the message key and the next chain key are computed using the symmetric-key ratchet [31]. The message key K_i and the next chain key k_{i+1} are computed using a Key Derivation Function (KDF) as

$$\begin{aligned} K_i &= H'(k_i) = \mathrm{KDF}(k_i, label_1, n_2) \ , \\ k_{i+1} &= H(k_i) = \mathrm{KDF}(k_i, label_2, n) \ . \end{aligned} \tag{1}$$

Shortly after the symmetric-key ratchet, the old chain key k_i is deleted, which gives forward secrecy. Compromise of k_{i+1} does not allow an attacker to calculate k_i. The Signal Protocol does not mandate any specific KDF and labels but recommends HMAC-SHA256 or HMAC-SHA512 and suggests $label_1$ = 0x01 and $label_2$ = 0x02. The Signal Protocol does not mandate any specific AEAD algorithm but recommends AES-256-CBC with HMAC-SHA256 or HMAC-SHA512. Irrespectively of the used algorithms, the size n of the chain keys and

the size n_2 of the message key are always 256 bits, i.e.,

$$n = n_2 = 256 \ . \tag{2}$$

Signal mandates that the symmetric-key ratchet is used for each message. When to use the Diffie-Hellman ratchet to derive a new initial chain key k_0 is left for the implementation. The Signal technical specification [31] does not give any recommendations or limits. Deriving a new initial chain key k_0 for each message or never deriving any new chain keys are both allowed according to the specification but the Double Ratchet algorithm is designed for quite frequent use of ephemeral Diffie-Hellman. Part of an example Double Ratchet key hierarchy is shown in Fig. 1.

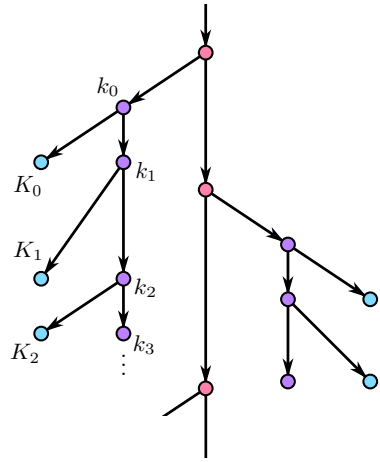

Fig. 1. Part of an example Double Ratchet key hierarchy.

2.2 TLS 1.3 and the Key Update Mechanism

TLS 1.3 [26] consists of a handshake protocol based on the theoretical SIGMA-I protocol [18] and a record protocol. After the TLS handshake is finished an initial traffic secret k_0 = application_traffic_secret_0 is derived (to simplify things we only discuss one of the directions). A chain of keys $k_0, k_1, k_2 \ldots$ derived from the initial traffic secret k_0 are used by the record protocol to protect all future messages sent in one direction over the connection including application data, post-handshake messages, and alerts. The size of the traffic secrets depends on the output size n of the hash function in the selected cipher suite. The five initial TLS 1.3 cipher suites registered by the TLS 1.3 specification [26] are listed in Table 1. As there are two senders (client and server) each connection has two

traffic secrets, one for each direction. For the rest of the connection, the keys in the two directions are independent of each other and in the rest of the paper we will only discuss one of the directions.

Once the handshake is complete, it is possible to update the traffic secret using the key update mechanism. The next traffic secret k_{i+1} is computed using a KDF based on HKDF-Expand [17] as

$$k_{i+1} = H(k_i) = \text{KDF}(k_i, \text{"traffic upd"}, n) \ . \tag{3}$$

Shortly after key update, the old traffic secret k_i is deleted, which gives forward secrecy. Compromise of k_{i+1} does not allow an attacker to calculate k_i. The TLS 1.3 record protocol only uses ciphers with an Authenticated Encryption with Associated Data (AEAD) interface. The AEAD key K_i and initialization vector IV_i are derived from k_i as

$$\begin{aligned} K_i &= \text{KDF}(k_i, \text{"key"}, n_2) \ , \\ IV_i &= \text{KDF}(k_i, \text{"iv"}, 96) \ . \end{aligned} \tag{4}$$

The AEAD nonce for each record is calculated as IV_i XOR S where S is the record sequence number. The size of the key K_i depends on the AEAD key length n_2 in the selected cipher suite and is not equal to n as in the Signal Protocol. The size of the nonce is 96 bits for all the cipher suites listed in Table 1. The 64-bit sequence number S is initially set to 0, increased for each message, and then reset to 0 every time the key update mechanism is used.

Table 1. The five initial cipher suites in TLS 1.3 [26]

Cipher suite	n	n_2
TLS_AES_128_GCM_SHA256	256	128
TLS_AES_256_GCM_SHA384	384	256
TLS_CHACHA20_POLY1305_SHA256	256	256
TLS_AES_128_CCM_SHA256	256	128
TLS_AES_128_CCM_8_SHA256	256	128

A single AEAD key K_i is typically used to protect many record protocol messages. For each cipher suite, TLS 1.3 has a limit for the number of encryption queries q. Key update is recommended before the limit is reached (every $2^{24.5}$ records for AES-GCM), see Section 5.5 of [26]. Frequent use of the key update mechanism is therefore expected in connections where a large amount of data is transferred. TLS 1.3 does not restrict the number of key updates.

DTLS 1.3 Datagram Transport Layer Security (DTLS) 1.3 [28] is a datagram security protocol that uses the TLS 1.3 handshake and cipher suites. The only

change to the key update mechanism is that DTLS 1.3 restricts the number of key updates to 2^{48}. DTLS 1.3 also increases the requirements on key usage limits to apply to both the sending and receiving side, i.e., key update is recommended based on both the number of encryption queries q and the number of failed decryption queries v.

QUIC QUIC [16] is a general-purpose transport layer protocol with built in security used in e.g., HTTP/3. QUIC uses the TLS 1.3 handshake and cipher suites. Key update and key derivation are done in the same way as Eqs. (3) and (4) but with the labels "quic ku", "quic key", and "quic iv" and that both directions always do a key update at the same time instead of independently as in TLS 1.3 and DTLS 1.3. QUIC does not restrict the number of key updates. QUIC has similar key usage limits and requirements as DTLS 1.3.

3 Hidden Stream Ciphers and TMTO Attacks

3.1 Synchronous Stream Ciphers

As described in e.g., [19] the keystream z_i in a synchronous stream cipher depends only on the initial state σ_0 and the position i but is independent of the plaintexts p and the ciphertexts c. The output cycle of a synchronous stream cipher can be described by the equations

$$\begin{aligned} \sigma_{i+1} &= f(\sigma_i) \ , \\ z_i &= g(\sigma_i) \ , \\ c_i &= h(z_i, p_i) \ , \end{aligned} \tag{5}$$

where σ_0 is the initial state, f is the next-state (or update) function, g is the output function, and h is the function used to combine the keystream with the plaintext. In a binary additive stream cipher the function h is the exclusive or function (XOR). The schematic can be seen in Fig. 2.

It turns out that the symmetric-key ratchet in Signal [31] and the key update mechanism in TLS 1.3 [26] can be modeled as such (non-additive) synchronous stream ciphers. The initial internal state is k_0, the next-state function $k_{i+1} = H(k_i)$ modifies the inner state, the output function $z_i = (K_i, IV_i) = g(k_i)$ uses the inner state to produce "keystream" z_0, z_1, ..., and the ciphertexts are a function $c_i = h(z_i, p_i)$ of "keystream" and plaintext, where p_i is all the application data encrypted with the key K_i.

3.2 Time Memory Trade-Off Attacks

Stream ciphers with internal states are vulnerable to Time Memory Trade-Off (TMTO) attacks. There are various TMTO attacks on synchronous stream ciphers such as Babbage-Golić [3] and Biryukov-Shamir [8]. These attacks take advantage

σ_i state $= k_i$
z_i keystream $= (K_i, IV_i)$

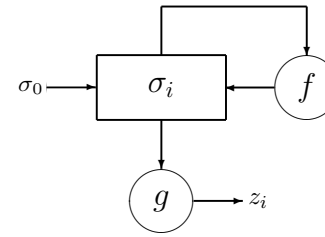

Fig. 2. Initiation and output cycle of a synchronous stream cipher.

of the internal state and apply to the Signal symmetric-key ratchet and the TLS 1.3 key update as well. TMTO attacks allow an attacker to find an internal state k_i from a set of output strings $y_0, y_1, \ldots, y_{D-1}$. When the state k_i is found, the attacker can derive all the future states $k_{i+1}, k_{i+2}, \ldots$, key material (K_i, IV_i), $(K_{i+1}, IV_{i+1}), \ldots$, and plaintexts $p_i, p_{i+1}, \ldots$ by running the keystream generator forward from the known state k_i. Both TMTO attacks are summarized in [8].

Babbage-Golić In Babbage-Golić [3], the attacker tries to find one of the many internal states instead of the key. The attacker generates M random states $k_0, k_1, \ldots, k_{M-1}$ from the total number of states N, calculates an output string y_j for each state k_j, and stores the pairs (k_j, y_j) ordered by y_j. In the real-time phase the attacker collects D output strings $y_0, y_1, \ldots, y_i, \ldots, y_{D-1}$. Requirements on the output strings are explained in Sect. 3.3. By the birthday paradox the attacker can find a collision $y_i = y_j$ and recover an inner state k_i in time $T = N/M$, memory M, data D, and preprocessing time $P = M$, where $1 \leq T \leq D$. Example points on this tradeoff relation is $P = T = M = D = N^{1/2}$, as well as and $T = D = N^{1/4}$ and $P = M = N^{3/4}$. This is very similar to a normal birthday attack where an attacker can recover a single key with the same complexities. The difference is that in the Babbage-Golić attack, the attacker, on average, recovers the last $D/2$ states $k_i, \ldots, k_{D-2}, k_{D-1}$ as well as any future states. If D is limited, a reasonable assessment (given that the attacker recovers $\approx D$ states) is that the security is reduced by

$$\min(d, n/2) \ , \tag{6}$$

where $d = \log_2 D$ and $n = \log_2 N$. If D is unlimited the security is reduced by $n/2$ bits when the attacker uses the tradeoff $P = T = M = D = N^{1/2}$.

Biryukov-Shamir In Hellman's attack on block ciphers [12], the attacker generates tables covering the N possible keys, but only stores the leftmost and rightmost columns in the table. Biryukov-Shamir [8] combines the Hellman and Babbage-Golić attacks. The attacker generates tables covering N/D states instead

of N keys in Hellman's attack [12]. In the real-time phase the attacker collects D output strings $y_0, y_1, \ldots, y_i, \ldots, y_{D-1}$ and can recover an inner state k_i in time $T = N^2/(M^2D^2)$ and preprocessing time $P = N/D$, where $D^2 \leq T \leq N$. Example points on this tradeoff relation is $P = T = N^{2/3}$ and $M = D = N^{1/3}$, as well as $P = N^{3/4}$, $D = N^{1/4}$, and $M = T = N^{1/2}$. Compared to Hellman's attack on block ciphers [12], Biryukov-Shamir's attack on stream ciphers runs D^2 times faster and the attacker, on average, recovers the last $D/2$ states $k_i, \ldots, k_{D-2}, k_{D-1}$ as well as any future states. If D is unlimited the security is reduced by $n/2$ bits.

3.3 TMTO Attacks on Signal and TLS 1.3

The effective stream cipher specific time memory trade-offs (TMTO) will be possible as long as the state size is less than twice the security level. As the name implies, the trade-off attacks give the attacker many possibilities. In addition to the discussion above, an attacker might also launch attacks where the probability of recovering a key is notably less than 1.

Based on these attacks, modern stream ciphers such as SNOW-V [10] follow the design principle that the security level is at most $n/2$ and that the state size in bits n should therefore be at least twice the security level. In his attack paper, Babbage [3] states that this principle is desirable. Zenner [34] states that a state size at least twice the security level is a necessary requirement for security. This is a reasonable requirement, especially if the number of key updates is unlimited.

The requirements on the output strings $y_0, y_1, \ldots, y_{D-1}$ depend on the function $h()$ used to combine the keystream with the plaintext $c_i = h(z_i, p_i)$. If $h()$ like AES-GCM, AES-CCM, and ChaCha20-Poly1305 is a combination of an additive stream cipher and a MAC the attack can be done with partially known and different plaintexts where y_i is a substring of $c_i \oplus p_i$. If $h()$ is AES-CBC, the attack requires that all the plaintexts have the same known prefix and y_i is a prefix of the ciphertext c_i. See Sect. 3.4 for a discussion on the practicality of the equal plaintext prefix model. The standard requirement today is that protocols should provide confidentiality against adaptive chosen ciphertext attacks.

TLS 1.3 and Signal do not explicitly state the intended security level, but the key length of the AEAD key can typically be seen as the intended security level. If we use the key length of the AEAD keys K_i as the security level, we see that TLS 1.3 and Signal do not follow design principles for stream ciphers. The reason for this is likely that the non-obvious stream cipher structure was overseen. The state size in Signal is always equal to the security level and the state size in TLS 1.3 is in some cases equal or 1.5 times the security level. As a result, TLS 1.3 and Signal offer far less than the expected security against these types of TMTO attacks.

3.4 Equal Plaintext Prefix

Being able to make stronger assumptions than that plaintexts are in English, Italian, German, or some other language can significantly improve cryptanalysis.

The cryptanalysis of Enigma ciphertext was e.g., improved by the assumption that certain German messages were likely to be the stereotypical phrase *"Keine besonderen Ereignisse"* or begin with the stereotypical prefix *"An die Gruppe"*. In the computer age we can almost always make such stronger assumptions. The application data sent over TLS is almost always using some protocol, which most likely has (known) fixed information fields such as headers. One of many examples is HTTP/1.1 [11] where the header for each request and response might begin with a lot of partly known data elements such as

```
GET /somewhere/fun/ HTTP/1.1
Host: www.example.com
User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l
Accept-Language: sv, tlh

HTTP/1.1 200 OK
Date: Thu, 12 August 2021 04:16:35 GMT
Server: Apache
Last-Modified: Mon, 5 August 2019 11:00:26 GMT
ETag: "34aa387-d-1568eb00"
Accept-Ranges: bytes
Content-Length: 51
Vary: Accept-Encoding
Content-Type: text/plain
```

Assuming partially known different plaintexts or that all the prefixes of the plaintexts are the same are very reasonable assumptions that are likely to apply in practice and can be used by an attacker. But note that protocols that do not provide confidentiality against adaptive chosen ciphertext attacks are typically to be considered broken.

4 Signal Protocol - Analysis and Recommendations

The Signal technical specification [31] does not aim for interoperability between different implementations and therefore has fewer details than the TLS 1.3 specification [26]. As the Signal protocol documentation does not give any recommendations or limits on how many times the symmetric-key ratchet can be used before the Diffie-Hellman ratchet is used we have to assume that in the worst case the symmetric-key ratchet can be used an unlimited number of times. We have not analyzed any implementations but even if attempts are made to transmit fresh ephemeral Diffie-Hellman keys as soon as possible, an attacker can hinder Diffie-Hellman to happen by blocking communication in one direction. In this case the Signal protocol gives a theoretical security level of 128 bits against TMTO attacks irrespectively of the used algorithms. This aligns with some of the recommended algorithms such as X25519 and SHA-256, but not other recommended algorithms such as X448, SHA-512, and AES-256, and not with the 256-bit key length of the message keys K_i.

A significant problem with the X3DH protocol [31] is that is does not mandate ephemeral Diffie-Hellman (as stated in the specification, the server might be all out of one-time Diffie-Hellman keys) and when ephemeral Diffie-Hellman is used the ephemeral Diffie-Hellman keys might be quite old. The X3DH specification [31] explains that this can be mitigated by quickly performing ephemeral Diffie-Hellman post-X3DH, but this is not mandated or even clearly recommended. The Double ratchet does not help as the initiating party can send messages before receiving an ephemeral public key from the responding party. Such messages provide neither forward secrecy with respect to long-term keys nor replay protection. Old ephemeral Diffie-Hellman keys are problematic as they are to be considered long term-keys and therefore cannot be used to provide *forward secrecy with respect to long-term keys*, which often is a desired property, promised for example by the TLS 1.3 handshake, see Appendix E.1 of [26].

As a first step we recommend that the Signal protocol documentation mandates a low limit on the number of times the symmetric-key ratchet can be used and gives clear security levels provided by the Signal protocol for different choices of algorithms. A limit on the number of times the symmetric-key ratchet can be used puts a limit on the data variable D, which following Eq. (6) improves the security level against TMTO attacks. With a low limit, the Signal protocol would provide a theoretical security level close to 256 bits when 256-bit algorithms are used, see Table 2.

We recommends that the Signal protocol documentation mandates quickly performing ephemeral Diffie-Hellman post-X3DH if the X3DH protocol did not include ephemeral Diffie-Hellman with recently generated keys. Before ephemeral Diffie-Hellman with fresh keys has been performed, the Initiator should restrict the type of messages that can be sent similar to zero round-trip time (0-RTT) data in TLS 1.3 [26], where HTTPS implementations typically only allow GET requests with no query parameters.

Mandating frequent use of ephemeral Diffie-Hellman also limits the impact of key compromise and forces an attacker to do dynamic exfiltration [5]. For IPsec, ANSSI [1] recommends enforcing periodic rekeying with ephemeral Diffie-Hellman every hour and every 100 GB of data, but we think that the Signal Protocol can and should have much stricter requirements than so. The impact of static key exfiltration with different rekeying mechanisms in TLS 1.3 is illustrated in Fig. 3. The symmetric-key ratchet in Signal has similar properties as the TLS 1.3 key_update and the Diffie-Hellman ratchet has similar properties as the TLS 1.3 rekeying with (EC)DHE.

We also recommend that the Signal protocol allows and recommends use of 512-bit chain keys together with the 256-bit message keys.

All analysis of the X3DH protocol applies also to the Post-Quantum Extended Diffie-Hellman (PQXDH) key agreement protocol [31], an upgrade to the X3DH specification released in September 2023 which makes the Signal protocol quantum-resistant.

Table 2. Security level as a function of D

N	D	Security level
2^{256}	∞	128
2^{256}	2^{64}	192
2^{256}	2^{32}	224
2^{256}	2^{16}	240
2^{256}	2^{0}	256

5 TLS 1.3 Family - Analysis and Recommendations

5.1 Time Memory Trade-Off Attacks

As TLS 1.3 [26] and QUIC [16] do not give any recommendations or limits on how many times key update can be used we have to assume that in the worst case the symmetric-key ratchet can be used an unlimited number of times (we have not analyzed any implementations). In this case TLS 1.3 and QUIC with TLS_-CHACHA20_POLY1305_SHA256 gives a theoretical security level of 128 bits against TMTO attacks and TLS_AES_256_GCM_SHA384 gives a maximum theoretical security level of 192 bits against TMTO attacks irrespectively of the used key exchange algorithm. This does not align with the 256-bit key length of the traffic secrets K_i. As stated in [24], the ChaCha20 cipher is designed to provide 256-bit security.

As DTLS 1.3 [28] restricts the number of key updates to 2^{48}, DTLS 1.3 with TLS_CHACHA20_POLY1305_SHA256 gives a theoretical security level of 208 bits, which does not align with the 256-bit key length of the traffic secrets K_i. Due to the restricted number of key updates, we assert that DTLS 1.3 with TLS_AES_256 _GCM_SHA384 gives 256 bits security if it is used with an equally secure key exchange algorithm.

As a first step we recommend that TLS 1.3 [26] and QUIC [16] mandate the same 2^{48} limit as DTLS 1.3 on the number of times a key update can be used and give clear security levels provided by different choices of algorithms. A limit on the number of key updates puts a limit on the data variable D, which following Eq. (6) improves the security level against TMTO attacks. With a 2^{48} limit, TLS 1.3 and QUIC would provide a theoretical security equal to the length of the traffic secrets K_i for all cipher suites except TLS_CHACHA20_-POLY1305_SHA256. Note that the cipher CHACHA20_POLY1305_SHA256 does give 256-bit security in TLS 1.3 when key update is not used. CHACHA20_-POLY1305_SHA256 also provides 256-bit security in TLS 1.2 when used with the rekeying mechanism renegotiation. We recommend that a new cipher suite TLS_CHACHA20_POLY1305_SHA512 is standardized for use with TLS 1.3.

TLS 1.3 should clearly state the intended security levels. We also recommend that TLS 1.3 mandates traffic secrets twice the AEAD key size for new cipher

suites. As an alternative, the transcript hash could be used as context in the key update instead of the empty context used today.

5.2 Key Exfiltration Attacks and Frequent Ephemeral Diffie-Hellman

Instances of large-scale monitoring attacks involving key exfiltration have been documented [15]. Moreover, it's highly probable that numerous additional occurrences have transpired clandestinely, escaping public acknowledgment. The avenues through which malicious entities can acquire keys are diverse, encompassing methods such as physical attacks, hacking, social engineering attacks, espionage, or by simply demanding access to keying material with or without a court order. Exfiltration attacks pose a significant and pressing cybersecurity threat [2].

The impact of static key exfiltration [5] with different rekeying mechanisms in TLS 1.3 is illustrated in Fig. 3. As can be seen the key update mechanism gives significantly worse protection against key exfiltration attacks than ECDHE. An attacker can perform a single static key exfiltration and then passively eavesdrop on all information sent over the connection even if the key update mechanism is used. With frequent ephemeral key exchange such as ECDHE, an attacker is forced to do active man-in-the-middle attacks or to do dynamic key exfiltration, which significantly increases the risk of discovery for the attacker [5]. The cost and risk associated with discovery is intricately tied to deployment specifics and the nature of the employed attack. In instances of a compromised system, automating key exfiltration could normalize costs between static and dynamic approaches. However, an augmented risk still stems from increased amounts of traffic volumes and log entries. Contrarily, in attack scenarios like side-channel attacks on Internet of Things (IoT) devices mandating physical proximity, the distinction between static and dynamic key exfiltration is substantial — encompassing both cost implications and the risk of discovery.

Two essential zero trust principles are to assume that breach is inevitable or has likely already occurred [23], and to minimize impact when breach occur [22]. One type of breach is key compromise or key exfiltration. As the key update mechanism gives significantly worse protection against key exfiltration attacks than ECDHE, TLS 1.3, DTLS 1.3, and QUIC should mandate frequent use of ephemeral Diffie-Hellman. For IPsec, ANSSI [1] recommends enforcing periodic rekeying with ephemeral Diffie-Hellman every hour and every 100 GB of data, we recommend the TLS 1.3 handshake to recommend this for non-constrained implementations. Constrained implementations should also mandate periodic rekeying with ephemeral Diffie-Hellman but could have a maximum period of 1 day, 1 week, or 1 month depending on how constrained the device and the radio is.

From what we can gather from IETF mailing lists, the standardization of TLS 1.3 might have placed too much emphasis on forward secrecy, possibly overlooking the significance of the additional security properties offered by frequent ephemeral key exchanges. In addition to ephemeral key exchange during a connection, TLS 1.3 also removed the possibility to perform post-handshake server authentication.

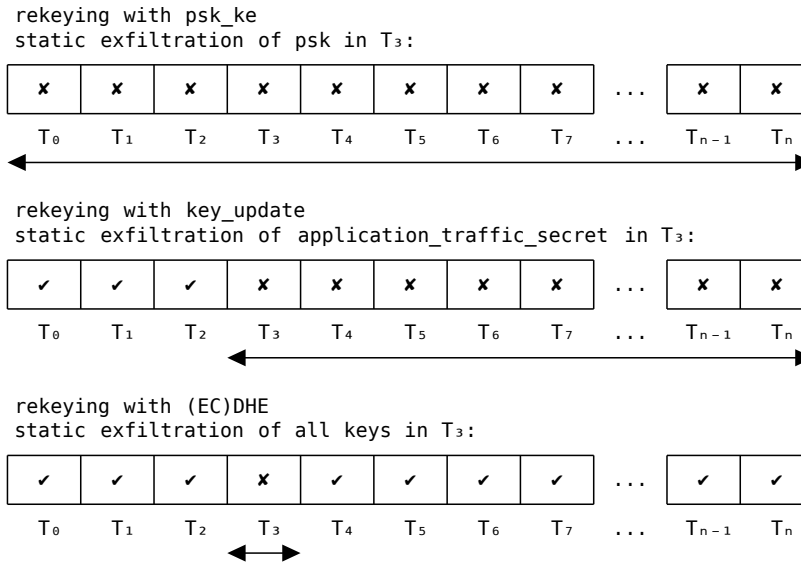

Fig. 3. TLS 1.3 - Impact of static key exfiltration in time period T_3 when psk_ke, key_update, and (EC)DHE are used.

The implications are that TLS 1.3, DTLS 1.3, and QUIC are unsuitable for long-lived connections and that protocols like DTLS/SCTP have to be redesigned to be able to frequently set up new connections. The upcoming revisions of the TLS 1.3 protocol and DTLS/SCTP have already been updated with descriptions and recommendations for frequent use of ephemeral Diffie-Hellman based on this work. See Appendix F.1 of [27] and Sections 3.4 and 9.1 of [33].

5.3 Analysis of the Procedure used to Calculate AEAD Limits

As specified in the TLS 1.3 and DTLS 1.3 specifications, implementations should do a key update before reaching the limits given in Section 5.5 of [26] and Section 4.5.3 of [16]. In QUIC key update must be done before the limits in Section 6.6 of [16] have been reached.

In TLS 1.3 the limits are just given without much further explanation. In DTLS 1.3 and QUIC procedures used to calculate the rekeying limits given in Appendix B of [28] and [16]. The DTLS 1.3 procedure specified in Appendix B of [28] suggest rekeying when the single-key confidentiality advantage (IND-CPA) is greater than 2^{-60} or when the single-key integrity advantage (IND-CTXT) is greater than 2^{-57}. QUIC has a similar procedure.

Our analysis is that these procedures are flawed both theoretical and in practice. The procedures uses single-key advantages to suggest rekeying which transform the problem to a multi-key problem and invalidates the single-key

calculation used to suggest the rekeying. Doing rekeying too early before the confidentiality or integrity of the algorithm decreases significantly faster than linear lowers the practical security and can create denial-of-service problems. The exact multi-key advantage depends on the algorithm but could be as much as m times its single-key advantage where m is the number of keys [6]. Multi-key advantages for the use of AES-GCM in TLS 1.3 is given by [6,13], which concludes that the nonce randomization do improve multi-key security for AES-GCM. We note that the nonce randomization do not improve security for ChaCha20-Poly1305 as $n = n_2$ and the 256-bit key K_i and the 96-bit IV are both derived from a 256-bit key k_i without any additional entropy. CHACHA20_POLY1305_-SHA256 was suitable for TLS 1.2 but is not suitable for TLS 1.3. Requiring rekeying after a low number of forgery attempts might be a denial-of-service problem as an attacker can affect availability with a small number of forgeries.

In general, an algorithm with a confidentiality advantage that is linear in the number of encryption queries q, e.g., $CA = q/2^{97}$, and with an integrity advantage that is linear in the number of failed decryption queries v, e.g., $IA = v/2^{103}$, does not need rekeying because of the advantages. But as explained in Sect. 5.2, rekeying is beneficial to limit the impact of a key compromise.

The confidentiality rekeying limits for AES-GCM [26] and AES-CCM [28] and the integrity rekeying limit for AES-CCM [28] coincides pretty well with when the confidentiality and integrity advantages starts to grow significantly faster than linear. These rekeying limits do significantly improve security. We do not know if this was luck or if the magic numbers 2^{-60} and 2^{-57} were chosen to achieve this.

The integrity limits for AES-GCM and ChaCha20-Poly1305 do not improve security as the single-key integrity advantages are bounded by a function linear in v, the number of forgery attempts. The forgery probability is therefore independent of the rekeying. Rekeying likely lowers the multi-key security but is unlikely to happen in practice as the limits are 2^{36} forgery attempts.

For CCM_8 the procedure gives illogical results unsuitable for practical use. Looking at the bound for the CCM_8 integrity advantage it is easy to see that CCM_8 performs very close to an ideal MAC for quite large number of failed decryption queries v. CCM_8 in itself is not a security problem for use cases such as media encryption or the Internet of Things, but the recommendations in [28] and [16] for CCM_8 are significant security problems as they introduce a denial-of-service problem, lowers security against TMTO attacks, and likely lowers the multi-key security. The denial-of-service problem comes from the DTLS 1.3 procedure recommending rekeying after 128 forgery attempts instead of the correct value $v \approx 2^{36}$ when the CCM_8 integrity advantage starts to grow significantly faster than linear. Applying the procedure on an ideal MAC with tag length 64 bits, i.e., an algorithm with integrity advantage $v/2^{64}$, gives the same illogical result, that the ideal MAC should be rekeyed extremely often.

While the rekeying recommendations for CCM_8 are illogical, we do agree with the decision to make CCM_8 with its 64-bit tags not recommended for general usage. For constrained IoT, we do however not see any practical problems

whatsoever. To have a 50% change of a single forgery, an attacker would need to send one billion packets per second for 300 years. This is completely unfeasible for constrained radio systems and the chance of this happening is negligible compared to the risk of data corruption due to hardware failure or cosmic rays.

We suggest that the procedures in Appendix B of [28] and [16] are deprecated in future versions. If any future procedure is needed it should be based on security per packet/byte/time instead of the practically irrelevant measures security per key/connection. Keeping some limit low per key or connection and then suggest rekeying or setting up a new connection will not increase practical security. If no good procedure can be found it is much better to just state limits as was done in [26], that is at least not wrong.

6 Conclusions, Recommendations, and Future Work

While we do not believe that the TMTO attacks pose a practical attack vector today, the attacks points to a fundamental design flaw in the key update mechanisms in TLS 1.3 and Signal, alternatively a lack of clearly stated security levels.

We find the design of the Signal protocol with a symmetric-key ratchet combined with a Diffie-Hellman ratchet very appealing as the protocol seems designed for frequent use of ephemeral Diffie-Hellman. It is possible that actual implementations already have hard limits on the number of times the symmetric-key ratchet can be used, meaning that they do provide close to 256-bit security and follows best practice when it comes to limit the impact of a key compromise.

We find several of the design choices in the TLS 1.3 handshake non-optimal resulting in that TLS 1.3 is problematic to use as a drop-in replacement of TLS 1.2. The standardization of TLS 1.3 might have placed too much emphasis on forward secrecy, possibly overlooking the significance of the additional security properties offered by frequent ephemeral key exchanges. Renegotiation was essential for frequent re-authentication and rekeying with ECDHE in DTLS/SCTP and the fourth flight in TLS 1.2 was essential for EAP-TLS. These problems can be overcome by using application data as a fourth flight [25] and by setting up new connections instead of using renegotiation [32].

Based on the analysis we recommend the Signal Protocol to:

- Introduce strict limits on the use of the symmetric-key ratchet.
- Mandate frequent use of the Diffie-Hellman ratchet based on time and data.
- Mandate ephemeral Diffie-Hellman with fresh keys before sending messages.
- Allow and recommend use of 512-bit chain keys.
- Clearly state the intended security level.

Based on the analysis we recommend TLS 1.3, DTLS 1.3, and QUIC to:

- Introduce strict limits on the use of the key update mechanism.
- Mandate frequent rekeying with EC(DHE) based on time and data.

- Standardize TLS_CHACHA20_POLY1305_SHA512.
- Mandate traffic secrets twice the AEAD key size for new cipher suites.
- Deprecate the procedure used for DTLS 1.3 and QUIC to calculate key limits.
- Clearly state the intended security levels.

Suggested future work:

- Evaluate the impact of this work on other protocols using symmetric ratchets such as MLS [4], EDHOC [29], and Key Update for OSCORE [14, 30] which have recently been standardized or are currently undergoing standardization.
- Evaluate implementations and deployments of the protocols. There are often significant differences between a specification, implementations of the specification, and actual deployments. One important aspect is to investigate would be how often actual deployments perform symmetric key update and ephemeral Diffie-Hellman and if an active attacker can influence the frequency.
- Provide guidance on how to use confidentiality advantages and integrity advantages for rekeying in security protocols.

Acknowledgements

The authors would like to thank Richard Barnes, Patrik Ekdahl, Loïc Ferreira, Alexander Maximov, Eric Rescorla, Ben Smeets, Erik Thormarker, and other reviewers for their helpful comments and suggestions.

References

1. Agence nationale de la sécurité des systèmes d'information: Recommendations for securing networks with ipsec (August 2015), https://www.ssi.gouv.fr/uploads/2015/09/NT_IPsec_EN.pdf
2. APNIC: How to: Detect and prevent common data exfiltration attacks, https://blog.apnic.net/2022/03/31/how-to-detect-and-prevent-common-data-exfiltration-attacks/
3. Babbage, S.: Improved "exhaustive search" attacks on stream ciphers. In: European Convention on Security and Detection, 1995. pp. 161–166 (1995). https://doi.org/10.1049/cp:19950490
4. Barnes, R., Beurdouche, B., Robert, R., Millican, J., Omara, E., Cohn-Gordon, K.: The Messaging Layer Security (MLS) Protocol. RFC 9420 (Jul 2023). https://doi.org/10.17487/RFC9420
5. Barnes, R., Schneier, B., Jennings, C.F., Hardie, T., Trammell, B., Huitema, C., Borkmann, D.: Confidentiality in the Face of Pervasive Surveillance: A Threat Model and Problem Statement. RFC 7624 (Aug 2015). https://doi.org/10.17487/RFC7624
6. Bellare, M., Tackmann, B.: The multi-user security of authenticated encryption: AES-GCM in TLS 1.3. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology – CRYPTO 2016, Part I. Lecture Notes in Computer Science, vol. 9814, pp. 247–276. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2016). https://doi.org/10.1007/978-3-662-53018-4_10

7. Bienstock, A., Fairoze, J., Garg, S., Mukherjee, P., Raghuraman, S.: A more complete analysis of the signal double ratchet algorithm. Cryptology ePrint Archive, Report 2022/355 (2022), https://eprint.iacr.org/2022/355
8. Biryukov, A., Shamir, A.: Cryptanalytic time/memory/data tradeoffs for stream ciphers. In: Okamoto, T. (ed.) Advances in Cryptology – ASIACRYPT 2000. Lecture Notes in Computer Science, vol. 1976, pp. 1–13. Springer, Heidelberg, Germany, Kyoto, Japan (Dec 3–7, 2000). https://doi.org/10.1007/3-540-44448-3_1
9. Cohn-Gordon, K., Cremers, C., Dowling, B., Garratt, L., Stebila, D.: A formal security analysis of the signal messaging protocol. Cryptology ePrint Archive, Report 2016/1013 (2016), https://eprint.iacr.org/2016/1013
10. Ekdahl, P., Johansson, T., Maximov, A., Yang, J.: SNOW-vi: an extreme performance variant of SNOW-V for lower grade CPUs. Cryptology ePrint Archive, Report 2021/236 (2021), https://eprint.iacr.org/2021/236
11. Fielding, R.T., Nottingham, M., Reschke, J.: HTTP Semantics. RFC 9110 (Jun 2022). https://doi.org/10.17487/RFC9110
12. Hellman, M.: A cryptanalytic time-memory trade-off. IEEE Transactions on Information Theory **26**(4), 401–406 (1980), https://ee.stanford.edu/~hellman/publications/36.pdf
13. Hoang, V.T., Tessaro, S., Thiruvengadam, A.: The multi-user security of GCM, revisited: Tight bounds for nonce randomization. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018: 25th Conference on Computer and Communications Security. pp. 1429–1440. ACM Press, Toronto, ON, Canada (Oct 15–19, 2018). https://doi.org/10.1145/3243734.3243816
14. Höglund, R., Tiloca, M.: Key Update for OSCORE (KUDOS). Internet-Draft draft-ietf-core-oscore-key-update-06, Internet Engineering Task Force (Oct 2023), https://datatracker.ietf.org/doc/draft-ietf-core-oscore-key-update/06/, work in Progress
15. Intercept, T.: How spies stole the keys to the encryption castle, https://theintercept.com/2015/02/19/great-sim-heist/
16. Iyengar, J., Thomson, M.: QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000 (May 2021). https://doi.org/10.17487/RFC9000
17. Krawczyk, D.H., Eronen, P.: HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869 (May 2010). https://doi.org/10.17487/RFC5869
18. Krawczyk, H.: SIGMA: The "SIGn-and-MAc" approach to authenticated Diffie-Hellman and its use in the IKE protocols. In: Boneh, D. (ed.) Advances in Cryptology – CRYPTO 2003. Lecture Notes in Computer Science, vol. 2729, pp. 400–425. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2003). https://doi.org/10.1007/978-3-540-45146-4_24
19. Mattsson, J.: Stream Cipher Design - An evaluation of the eSTREAM candidate Polar Bear. Master's thesis, Royal Institute of Technology (2006), https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.108.40
20. McGrew, D., Rescorla, E.: Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP). RFC 5764 (May 2010). https://doi.org/10.17487/RFC5764
21. McKay, K., Cooper, D.: Guidelines for the selection, configuration, and use of transport layer security (tls) implementations (August 2019). https://doi.org/10.6028/NIST.SP.800-52r2
22. National Institute of Standards and Technology: Implementing a zero trust architecture (July 2023), https://www.nccoe.nist.gov/sites/default/files/2023-07/zta-nist-sp-1800-35b-preliminary-draft-3.pdf

23. National Security Agency: Embracing a zero trust security model (February 2021), https://media.defense.gov/2021/Feb/25/2002588479/-1/-1/0/CSI_EMBRACING_ZT_SECURITY_MODEL_UOO115131-21.PDF
24. Nir, Y., Langley, A.: ChaCha20 and Poly1305 for IETF Protocols. RFC 8439 (Jun 2018). https://doi.org/10.17487/RFC8439
25. Preuß Mattsson, J., Sethi, M.: EAP-TLS 1.3: Using the Extensible Authentication Protocol with TLS 1.3. RFC 9190 (Feb 2022). https://doi.org/10.17487/RFC9190
26. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Aug 2018). https://doi.org/10.17487/RFC8446
27. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. Internet-Draft draft-ietf-tls-rfc8446bis-09, Internet Engineering Task Force (Jul 2023), https://datatracker.ietf.org/doc/draft-ietf-tls-rfc8446bis/09/, work in Progress
28. Rescorla, E., Tschofenig, H., Modadugu, N.: The Datagram Transport Layer Security (DTLS) Protocol Version 1.3. RFC 9147 (Apr 2022). https://doi.org/10.17487/RFC9147
29. Selander, G., Mattsson, J.P., Palombini, F.: Ephemeral Diffie-Hellman Over COSE (EDHOC). Internet-Draft draft-ietf-lake-edhoc-22, Internet Engineering Task Force (Aug 2023), https://datatracker.ietf.org/doc/draft-ietf-lake-edhoc/22/, work in Progress
30. Selander, G., Preuß Mattsson, J., Palombini, F., Seitz, L.: Object Security for Constrained RESTful Environments (OSCORE). RFC 8613 (Jul 2019). https://doi.org/10.17487/RFC8613
31. Signal: Signal technical documentation, https://signal.org/docs/
32. Tüxen, M., Rescorla, E., Seggelmann, R.: Datagram Transport Layer Security (DTLS) for Stream Control Transmission Protocol (SCTP). RFC 6083 (Jan 2011). https://doi.org/10.17487/RFC6083
33. Westerlund, M., Mattsson, J.P., Porfiri, C.: Datagram Transport Layer Security (DTLS) over Stream Control Transmission Protocol (SCTP). Internet-Draft draft-ietf-tsvwg-dtls-over-sctp-bis-07, Internet Engineering Task Force (Oct 2023), https://datatracker.ietf.org/doc/draft-ietf-tsvwg-dtls-over-sctp-bis/07/, work in Progress
34. Zenner, E.: On the role of the inner state size in stream ciphers. Cryptology ePrint Archive, Report 2004/003 (2004), https://eprint.iacr.org/2004/003