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 modelled 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 offers 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, DTLS 1.3, QUIC, Signal, Secret-key Cryptography, Key Derivation, Ratchet, Key Chain, Stream Cipher, Cryptanalysis, 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 [1] 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 [2], EAP-TLS 1.3 [3], DTLS 1.3 [4], DTLS-SRTP [5], and DTLS/SCTP [6] 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 [7]. Two nodes that support TLS 1.3 will never negotiate the obsolete TLS 1.2.

The Signal protocol [8] 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 $k = H(k)$. We show that the key update function in TLS 1.3 and the

symmetric key ratchet in Signal can be modelled 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 offers 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. The upcoming revisions of the TLS 1.3 protocol [9] and DTLS/SCTP [10] have already been updated based on this work.

2 Preliminaries

2.1 Signal Protocol and the Symmetric-Key Ratchet

The Signal protocol [8] 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 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. 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. How to use associated data is left for the application.

Before each message is sent, the chain key and message key are updated using the symmetric-key ratchet [8]. The next chain key k_{i+1} and the next message key K_{i+1} are computed using a Key Derivation Function (KDF) as

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

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 $label_1 = 0\text{x}01$ and $label_2 = 0\text{x}02$. 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 is 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 application. The Signal technical specification [8] 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.

2.2 TLS 1.3 and the Key Update Mechanism

TLS 1.3 [1] consists of a handshake protocol based on the theoretical SIGMA-I protocol 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 [1] 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 [11] 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.

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 [1]. 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.

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

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

DTLS 1.3 Datagram Transport Layer Security (DTLS) 1.3 [4] 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 [2] 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 (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., [12] 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. 1.

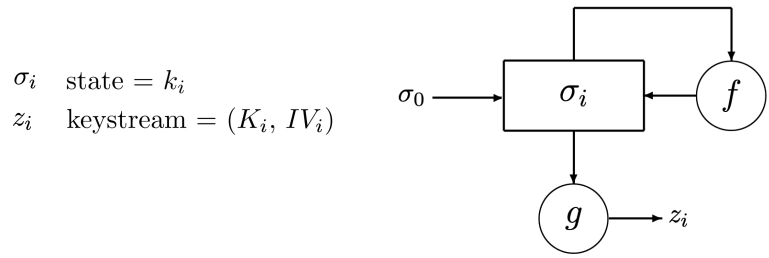

Fig. 1. Initiation and output cycle of a synchronous stream cipher.

It turns out that the symmetric-key ratchet in Signal [8] and the key update mechanism in TLS 1.3 [1] can be modelled 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 [14] and Biryukov-Shamir [13]. These attacks take advantage 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}, ... by running the keystream generator forward from the known state k_i. Both TMTO attacks are summarized in [13].

Babbage-Golić In Babbage-Golić [14], 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 Section 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 Biryukov-Shamir [13] the attacker generates tables covering N/D points instead of N points in Hellman's attack [15]. 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 [15], 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 [16] 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 [14] states that this principle is desirable. Zenner [17] 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 Section 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 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.

4 Signal Protocol - Analysis and Recommendations

As the Signal protocol documentation [8] 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). 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.

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 give 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 Equation (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.

Mandating frequent use of ephemeral Diffie-Hellman also limits the impact of key compromise and forces an attacker to do dynamic exfiltration [20]. For IPsec, ANSSI [18] 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. 2. 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.

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 [1] and QUIC [2] 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 [19], the ChaCha20 cipher is designed to provide 256-bit security.

As DTLS 1.3 [4] 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 [1] and QUIC [2] 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 Equation (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

The impact of static key exfiltration [20] with different rekeying mechanisms in TLS 1.3 is illustrated in Fig. 2. 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 send 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 dynamic key exfiltration, which significantly increases the risk of discovery for the attacker [20]. Two essential zero trust principles are to assume that breach is inevitable or has likely already occurred [21], 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 also

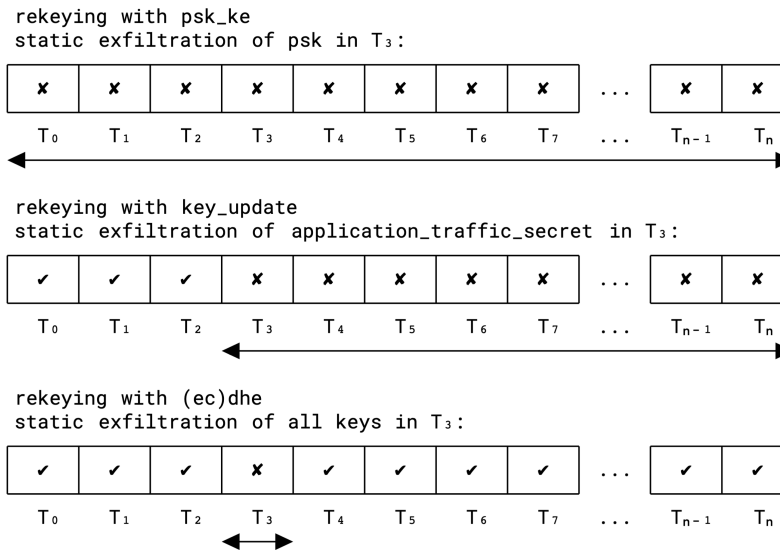

Fig. 2. TLS 1.3 - Impact of static key exfiltration in time period T_3 when psk_ke, key_update, and (ec)dhe are used.

mandate frequent use of ephemeral Diffie-Hellman. For IPsec, ANSSI [18] 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.

The standardization of TLS 1.3 might have been too focused on forward secrecy without realizing that frequent ephemeral key exchange provides more security properties than just forward secrecy. In addition to ephemeral key exchange during a connection, TLS 1.3 also removed the possibility to perform post-handshake server authentication. The implications are that TLS 1.3, DTLS 1.3, and QUIC are unsuitable for long-lived connections and that protocols like DTLS/SCTP has to be redesigned to be able to frequently set up new connections. The upcoming revisions of the TLS 1.3 protocol [9] and DTLS/SCTP [10] have already been updated based on this work.

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 [1] and Section

4.5.3 of [4]. In QUIC key update must be done before the limits in Section 6.6 of [2] 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 [4] and [2]. The DTLS 1.3 procedure specified in Appendix B of [4] 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.

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 Section 5.2, rekeying is beneficial to limit the impact of a key compromise.

The confidentiality rekeying limits for AES-GCM [1] and AES-CCM [4] and the integrity rekeying limit for AES-CCM [4] 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 just gives nonsense. 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 [4] and [2] 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 nonsense result, the ideal MAC should be rekeyed extremely often.

While the rekeying recommendations for CCM_8 are nonsense, 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 [4] and [2] 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 is theoretical nonsense and will not increase practical security. If no good procedure can be found it is much better to just state limits as was done in [1], that is at least not wrong.

6 Conclusions and Recommendations

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 been too focused on forward secrecy without realizing that frequent ephemeral key exchange provides more security properties than just forward secrecy. 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 [3] and by setting up new connections instead of using renegotiation [6].

Based on the analysis we recommend the Signal Protocol to:

- Introduce strict limits on the use of the symmetric-key ratchet.
- Enforce use of the Diffie-Hellman ratchet based on time and data.
- 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.

Acknowledgements

The authors would like to thank Patrik Ekdahl, Alexander Maximov, Ben Smeets, and Erik Thormarker for their helpful comments and suggestions.

References

1. IETF RFC 8446.: The Transport Layer Security (TLS) Protocol Version 1.3. August 2018 https://www.rfc-editor.org/rfc/rfc8446.html
2. IETF RFC 9000.: QUIC: A UDP-Based Multiplexed and Secure Transport. May 2021 https://www.rfc-editor.org/rfc/rfc9000.html
3. IETF RFC 9190.: EAP-TLS 1.3: Using the Extensible Authentication Protocol with TLS 1.3 February 2022 https://www.rfc-editor.org/rfc/rfc9190.html
4. IETF RFC 9147.: The Datagram Transport Layer Security (DTLS) Protocol Version 1.3. April 2022 https://www.rfc-editor.org/rfc/rfc9147.html
5. IETF RFC 5764.: Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP). May 2010 https://www.rfc-editor.org/rfc/rfc5764.html
6. IETF RFC 6083.: Datagram Transport Layer Security (DTLS) for Stream Control Transmission Protocol (SCTP). January 2011 https://www.rfc-editor.org/rfc/rfc6083.html
7. NIST SP 800-52r2.: Guidelines for the Selection, Configuration, and Use of Transport, Layer Security (TLS) Implementations. August 2019 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf
8. Signal Technical Documentation https://signal.org/docs/
9. IETF draft-ietf-tls-rfc8446bis.: The Transport Layer Security (TLS) Protocol Version 1.3. March 2023 https://datatracker.ietf.org/doc/html/draft-ietf-tls-rfc8446bis
10. IETF draft-ietf-tsvwg-dtls-over-sctp-bis.: Datagram Transport Layer Security (DTLS) for Stream Control Transmission Protocol (SCTP). April 2023 https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-dtls-over-sctp-bis
11. IETF RFC 5869.: HMAC-based Extract-and-Expand Key Derivation Function (HKDF). May 2010 https://datatracker.ietf.org/doc/rfc5869/
12. John Mattsson.: Stream Cipher Design. 2006 https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.108.40
13. Biryukov, Alex. Shamir, Adi.: Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. October 2000 https://link.springer.com/chapter/10.1007/3540-444483_1
14. Steve Babbage.: Improved "exhaustive search" attack on stream ciphers. June 1995 https://www.researchgate.net/publication/3625367_Improved_exhaustive_search_attacks_on_stream_ciphers
15. Martin Hellman.: A Cryptanalytic Time - Memory Trade-Off. July 1980 https://ee.stanford.edu/ hellman/publications/36.pdf

16. Ekdahl, Patrik. Johansson, Thomas. Maximov, Alexander. Yang, Jing.: SNOW-Vi: An Extreme Performance Variant of SNOW-V for Lower Grade CPUs 2021 https://eprint.iacr.org/2021/236
17. Erik Zenner.: On the Role of the Inner State Size in Stream Ciphers. January 2004 https://madoc.bib.uni-mannheim.de/724/1/zenner04a.pdf
18. 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
19. IETF RFC 8439.: ChaCha20 and Poly1305 for IETF Protocols. June 2018 https://datatracker.ietf.org/doc/rfc8439/
20. IETF RFC 7624.: Confidentiality in the Face of Pervasive Surveillance: A Threat Model and Problem Statement. August 2015 https://datatracker.ietf.org/doc/html/rfc7624
21. 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
22. National Institute of Standards and Technology.: Implementing a Zero Trust Architecture December 2022 https://www.nccoe.nist.gov/sites/default/files/2022-12/zta-nist-sp-1800-35b-preliminary-draft-2.pdf