Winter – Cloud Scale Cryptography Toolkit
[2022-08-06 © Stan Drapkin]
sdrapkin@sdprime.com

1st **Winter** primitive is a streaming chunked AEAD with CMT-4.

- AEAD is AEAD(K, N, A, M) where K=key, N=nonce, A=associated data, M=message
- CMT-1 is AEAD that commits to K only
- CMT-4 is AEAD that commits to all 4 (K, N, A, M)

GCM-based, with https://eprint.iacr.org/2022/268.pdf and https://eprint.iacr.org/2020/1456.pdf
AEAD (GCM) → CMT-1 (key-commitment tag) → CMT-4 (commitment to K, N, A, M)

GCM → CMT-1:

Algorithm 2: $\mathsf{CommitKey}_{IV}\Pi^{\mathsf{Enc}}(K, N', N, A, M)$

Input: $K \in \{0,1\}^{\kappa_0}$, $N' \in \{0,1\}^{\nu'}$, $N \in \mathsf{N}$, $A \in \mathsf{A}$, $M \in \mathsf{M}$
Output: $C \in \mathsf{C}$, $K_{\mathrm{com}} \in \{0,1\}^{c}$

$K_{\mathrm{enc}} \leftarrow F_{\mathrm{enc}}(K, N')$
$K_{\mathrm{com}} \leftarrow F_{\mathrm{com}}(K, N')$
$C \leftarrow \mathsf{Enc}(K_{\mathrm{enc}}, N, A, M)$
return (C, K_{com})

CMT-1 → CMT-4 (MB/VTH original scheme):

$\overline{\mathsf{SE}}.\mathsf{Enc}(K, N, A, M)$	$\overline{\mathsf{SE}}.\mathsf{Dec}(K, N, A, C^* \| T')$
$L \leftarrow H(K, (N, A))$	$L \leftarrow H(K, (N, A))$
$C \leftarrow \mathsf{SE.Enc}(L, N, \varepsilon, M)$	$M \leftarrow \mathsf{SE.Dec}(L, N, \varepsilon, C)$
Return C	Return M

Using a combined version for GCM → CMT-4 (with some tweaks):

Inputs: User Key [UK:*], Plaintext [PTX:*], optional Associated Data [AD:*], Chunk Length CL=constant

- HASH is SHA512
- HKDF is HKDF<HASH>
- GCM is GCM with 256-bit key and 128-bit tag
- [CN:8] is little-endian 8-byte UInt64

Encrypt per-Message:

1. [MID:40] ← Random Byte Generator RBG:40 (MID is Message ID)
2. [ADH:64] ← HASH([AD:*])
3. [KDK:64] ← HKDF.Extract(ikm: [UK:*], salt: ["KDK"][MID:40][ADH:64]) // salt not to exceed 128 bytes
4. [MCT:32] ← HKDF.Expand(prk: [KDK:64], len: 32, info: ["MCT"][MID:40][ADH:64]) // info not to exceed 109 bytes
5. [MHR:72] ← [MID:40][MCT:32] // MCT is Msg Commitment Tag
6. Output Message HeadeR [MHR:72]
7. Keep in internal per-message state: [KDK:64], [MHR:72]

Encrypt per-Chunk:

1. [CEK:32][CIV:12] ← HKDF.Expand(prk: [KDK:64], len: 44, info: ["CEI"][MHR:72][CN:8])
2. GCM.Enc(key:[CEK:32], iv:[CIV:12], ptx:[PTX:CL], ctx: out [CTX:CL], ad:[empty], tag: out [GT:16])
3. Return [CTX:CL][GT:16]

1st **Winter** primitive -- streaming chunked AEAD – output:

Header	Chunk 1	Chunk ...	Chunk L (Last)

- Header is [MID:40][MCT:32]
- Full Chunk is 128 KiB (1024*128=131,072 bytes)
  - [131,056 bytes of GCM ciphertext][16 bytes of GCM tag]
- Last Chunk is strictly < 131,072 bytes (Chunk 1 can be Last Chunk)
- Encrypted message can have up to 2^64 chunks
  - Maximum message size is 2 YiB (Yobibytes). 1 Yobibyte = 2^80 bytes.
- 2^128 Messages can be created with MID collision probability 1/2^64.
- Per-Chunk GCM-tags provide ~115 bits of security (NIST requires no fewer than 112 bits)..

Winter provides primitives for building Cloud Scale Encryption Systems (described by Campagna/Gueron 2019):

- Any (very large) number of users
- Any (very large) number of messages
- Any (very large) number of keys
- Any (very large) number of encryptions per key
- Any (very large) maximum message size

Winter – Cloud Scale Cryptography Toolkit discussion:

1st **Winter** primitive -- streaming chunked AEAD

1. Design motivation (why bother creating it?):
   - Adam Langley blog (Senior Staff Engineer @Google)
   - *"How do I encrypt lots of records with a per-user key?"* — (every) Anonymous developer
   - This symmetric-cryptography question is the original motivation of **ALL OF CRYPTOGRAPHY**.
   - You would have thought that we would have figured it out by now, but we totally haven't.
   - This was true in 2015 and is still true in 2022.
   - Cloud Scale cryptographic systems changed the definition of "lots".
2. Design goals (what are we aiming to achieve?):
   - Streaming chunked AEAD with commitment to all 4 inputs (Key, Nonce, Associated Data, Message)
   - Cloud Scale Cryptography – safe security bounds (beyond AES birthday bounds) under very large:
     - plaintext sizes, # of users/keys/operations per key/forgery attempts
   - Utmost safety & usability ("*Security at the expense of usability comes at the expense of security.*")
   - Regulatory Compliance Ready (targeting NIST as well as international compliance requirements)
     - *"The best cryptography is useless if it cannot be used."*
   - Implementation interoperability across modern languages/frameworks
     - .NET, Java, Golang, Python, Swift, JS, Rust, C/C++, PHP, etc.
     - (ex. sp800_108 KDFs are rarely available, but HKDF is ubiquitous)
   - Fast speed (throughput), targeting at least 3 GiB/second on 1 average CPU core
   - Encryption and Decryption speed should scale with additional CPUs/memory
   - Single ciphertext byte-format (regardless of CPUs/memory used, or whether streaming or not)
     - addresses usability requirement
   - Parallelizable chunk encryption, random chunk access (decryption)
   - Every byte of output should be (1) indistinguishable from random; (2) authenticated
   - Minimal plaintext-to-ciphertext size expansion
3. Cryptographic primitives used:
   - **AES-GCM** (NIST SP 800-38D) with 256-bit key and 128-bit tag
   - **SHA-512** (RFC 6234, FIPS 180-4)
   - **HKDF** (RFC 6234, NIST SP 800-56C, NIST ACVP), which uses HMAC (RFC 5869, FIPS 198-1)

Winter – Cloud Scale Cryptography Toolkit – FAQ

1. Why use it?
    a. **Extreme performance.** Early streaming-AEAD prototypes run at ~**6.5 GiB/second** on a 4-CPU Intel.
    b. Usable cryptography. No limits to worry about, no cryptographic-wearout of anything, no need to rekey.
    c. No choices/decisions to make. Only 1 protocol per primitive, no interoperability issues.
    d. Designed for regulatory compliance (safe to use in your Business/Government/Industry).
2. Who is designing it?
    a. Stan Drapkin is responsible for the design (and any mistakes), but design ideas were shaped by discussions with cryptography experts. Your feedback is appreciated.
3. Is it Production-ready?
    a. No (not yet), but at some point it will be. The 1st stage is to get as much feedback as possible.
4. Is implementation ready?
    a. If early feedback does not identify big issues, an MVP/reference implementation will be shared on Github.
5. Why is Chunk size 128 KiB?
    a. 128 KiB chunks achieve the best performance (throughput) in our tests, while balancing other factors.
6. How does **Winter** streaming AEAD compare with Google Tink? (not a comprehensive comparison)
    a. Tink provides crypto-agility. Winter has no cryptographic agility by design (only 1 protocol per primitive).
    b. Tink's AEADs (plural) use a same-key GCM for all chunks of a message (changing Nonce only).
    c. Tink's limits are inferior to Winter's AEAD limits.
    d. Tink's performance is inferior to Winter's AEAD performance.
    e. Tink has no official .NET support.
    f. Tink AEADs are not CMT-1 or CMT-4.
7. How does **Winter** streaming AEAD compare with AWS Encryption SDK? (not a comprehensive comparison)
    a. AESDK provides crypto-agility. Winter has no cryptographic agility by design (only 1 protocol per primitive).
    b. AESDK AEADs use a same-key GCM for all chunks of a message (changing Nonce only).
    c. AESDK limits are inferior to Winter's AEAD limits.
    d. AESDK performance is inferior to Winter's AEAD performance.
    e. AESDK does not support streaming for .NET.
    f. AESDK AEADs are CMT-1 but may not be CMT-4.