|galois|

Galois Embedded Crypto: Light Weight Cryptography

Thomas M. DuBuisson <tommd@galois.com>

15 April 2015

Introduction

Galois Embedded Crypto (GEC) is a light-weight communication encryption system suitable for devices with low bandwidth, computation and limited program memory. Resource-constrained devices, such as sensor systems and unmanned vehicles, have critical need of secure communications but the publicly available mechanisms are in the form of all-encompassing cryptographic libraries, special hardware modules, and individual algorithms. The consequence is that many devices omit any cryptographic protections or use home-grown solutions that have fundamental weaknesses. GEC seeks to fill this niche with a well-specified protocol and matching implementation.

SMAVLink, the GEC predecessor, provided secure communications between parties who held a single-use symmetric key. While conceptually sound, this put an undue burden on the user to regularly distribute cryptographic keys through an out-of-band mechanism (read: re-flashing a quad-copter's firmware before every flight). GEC uses the same peer-reviewed and red-teamed design as SMAVLink, but adds key exchange using elliptical curve cryptography to provide a more sensible work flow that encourages proper use of the cryptographic system.

In this document I first discuss the GEC protocol then its implementation. The protocol includes the cryptographic design as well as the wire-format. The implementation section is a brief summary of the implemented components and borrowed cryptographic implementations.

GEC Protocol

GEC is loosely based on the station-to-station key exchange protocol followed by ongoing communications secured with traditional symmetric-key cryptography. These phases are described below both in terms of cryptographic and on-the-wire formats.

Throughout the description I use variable P for public keys, Q for private keys, K for symmetric keys, S for salt values, and subscripts to denote data owners. The operator `||` is concatenation. $E()$ is AES encryption using either counter mode (key exchange) or AES GCM (ongoing communications).

Key Exchange

GEC key exchange is accomplished by combining the Station-To-Station (STS) protocol with curve25519 for key exchange, ed25519 for authentication, SHA512 with a counter for key derivation, and finally AES128 (counter mode) for the key confirmation. Assume parties A and B with ed25519 asymmetric key pairs (P_a,Q_a) and (P_b,Q_b) The protocol proceeds as:

1. A generates an ephemeral (random) curve25519 key pair (P_{ae}, Q_{ae}) and sends P_{ae}.

2. B generates ephemeral curve25519 key pair (P_{be}, Q_{be}).
3. B computes the shared secret: z = scalar_multiplication(Q_{be}, P_{ae})
4. B uses the key derivation function kdf(z,1) to compute K_b || S_b, `kdf(z,0)` to compute K_a || S_a, and `kdf(z,2)` to compute K_{client} || S_{client}.
5. B computes the ed25519 signature: sig = $sign_{Qb}$(P_{be} || P_{ae})
6. B computes and sends the message P_{be} || $E_{key=Kb,IV=Sb||zero}$(sig)
7. A computes the shared secret: z = scalar_multiplication(Q_{be}, P_{ae})
8. A uses the key derivation function kdf(z,1) to compute K_b || S_b, `kdf(z,0)` to compute K_a || S_a, and `kdf(z,2)` to compute K_{client} || S_{client}.
9. A decrypts the remainder of the message, verifies the signature.
10. A computes the ed25519 signature: sig = $sign_{Qa}$(P_{ae} || P_{be})
11. A computes and sends the message $E_{key=Ka,IV=Sa||zero}$(sig)
12. A returns the values (K_{client},S_{client}) to the callee as the resulting key material.
13. B decrypts the message and verifies the signature. B then returns the tuple (K_{client},S_{client}) to the callee as the resulting key material.

The wire-format is unsurprisingly a reproduction of the messages in the above computations:

- message1 = [P_{ae} (32 bytes)]
- message2 = [P_{be} (32 bytes) | Encrypted Signature (64 bytes)]
- message3 = [Encrypted Signature (64 bytes)]

Key Derivation The key derivation function is a SHA512 hash of the concatenation of a 16 bit big endian counter, the shared secret 'z', and a one byte party-specific identifier (0 for A, 1 for B and 2 for key material returned to the callee). The requested sizes are all under the output size of SHA512 so the counter is always zero and the return values are the first N bytes of the hash depending on the amount requested.

```
kdf(z,partyIdent) = SHA512( 0 || z || partyIdent)
```

N.B. This key derivation technique is a common one, appearing in NIST SP 800-56A and other standards.

Costs For each party, key exchange costs 96 bytes of bandwidth (1 and 2 transmission depending on the role of the party), 32 bytes of random values, two scalar multiplications (to obtain the ephemeral public key and to compute the shared secret), one signature verification, one signature generation, 8 blocks of AES-CTR, and three SHA512 hashes over 34 bytes each (producing 96 bytes of key material).

These operations are close to minimal for the protocol in question and are believed to be cheap enough for all targeted uses. Some micro-optimizations could be made to (for example) run 2 SHA512 operations instead of 3, however a first glance suggests that any significant performance improvements would have to come from taking advantage of available hardware accelerations or selection of faster algorithms; these decisions usually involve a trade-off between the portability, engineering effort, or even security margin of the system.

|galois|

Analysis The STS protocol is simpler than, but not as featureful as, standardized protocols such as TLS or Internet Key Exchange (IKE). The simplicity allows for a smaller code base that benefits audit-ability as well as memory foot-print. On the down-side, key lifetimes and re-negotiation becomes the responsibility of the consumer. Additionally, party B is at a significant disadvantage with respect to resisting resource-exhaustion attacks due to the large amount of up-front work caused by a cheap, 32 byte, anonymous message.

Some of these short-comings can be overcome with relatively small changes to the protocol and little impact in the memory footprint. However, the complexity costs must be weighed against any need for such feature, particularly in the face of other insecurities in the overall system such as wireless spectrum exhaustion and other lower-level attacks.

Security of Ongoing Communications

Once the symmetric keys have been established, ongoing communications are secured using a counter and AES in Galois Counter Mode (GCM). The counter exists both to provide a unique initialization vector (IV) per message and to allow elimination of old or duplicate messages, thus providing resistance to replay attacks. Given unique IVs and secret keys, the AES GCM algorithm provides confidentiality and integrity of messages.

Outgoing Messages Remember the key exchange protocol yielded 24 bytes in the form of K_c (16 bytes) and S_c (8 bytes). For each message a unique IV is constructed by concatenating the salt and 32 bit counter: S_c || counter. The counter is initially zero and incremented before processing each message. The ciphertext and tag are computed as ciphertext || tag = $E_{Key=Kc,IV=Sc||counter}(message)$ and sent as the datagram: counter (32 bit, bit endian) || ciphertext || tag (first 96 bits).

Incoming Messages The context for incoming messages is the zeroed counter, K_c, and S_c. Each received message is first checked to ensure the included counter value is greater than the recorded count, then the AES GCM routine is used to verify the tag and decypt the message. If the GCM tag is valid the stored counter is updated with the value from the current message prior to returning the plaintext.

Analysis GEC uses a 96 bit authentication tag while GCM tags have a full length of 128 bits. The decision to use small tage size is motivated by the bandwidth constraints inherent in our domain; every message of 80 bytes must be placed in a 96 byte packet. NIST and independent recommendations are to remain under 2^32 messages when using 64 bit tags - micro air vehicles average 100 packets per second while mission lifetime is under a day, putting us well below this bound.

The counter is 32 bits which, when combined with the salt, results in the recommended GCM IV length of 96 bits. A consequence of 32 bit counters is a limit of 2^32 messages, which is in line with the GCM tag size. Notice that these limits are somewhat misleading - if the useful life of a symmetric key is exhausted it is possible to renegotiate a new key using the key exchange protocol.

On ingress, recording only the largest observed counter value (as described above) can be non-optimal for some uses. It is a trivial change to have a moving bit-window recording the observed counters from some offset. This popular technique allows out-of-order messages without exposing the system to replay attacks and its use is encouraged any time messages are sent over the Internet.

GEC Implementation

GEC is an accumulation of well-known implementations of the core cryptographic algorithms carefully used to perform key exchange and secure the steady-state communications. The algorithm implementations include:

- Curve25519-donna from Adam Langley
- Ed25519-donna from Andy Moon
- GladmanAES and GCM from Brian Gladman

The integrating GEC Code is split across two files. The file `gec.c` is a unifying C wrapper that provides types, key initialization and generation, signing, verification, encryption, and decryption operations along with the code for post key-exchange communications via `gec_encrypt` and `gec_decrypt`. The `gec-ke.c` file provides a high level interface of producers and consumers of the three messages via four primary functions: `initiate_sts`, `respond_sts`, `resposne_ack_sts`, and `finish_sts`.

Conclusions

I have presented GEC, an evolution of the previous SMACCM Secure MAVLink system that includes key exchange and cleaner IV construction. The design is believed to be a more user-friendly solution due to simpler key management while providing a stronger security posture by avoiding frequent key sharing between numerous parties. The current system uses AES in order to leverage previous work, prior peer-review, and conform to reasonable standards; I acknowledge switching to ChaCha20 could be a sensible future step although it strongly depends on the importance of standards and availability of algorithm-specific hardware acceleration for the use case in question. In the case of key exchange the recently developed algorithms curve25519 and ed25519 were chosen due to the clear benefits over NIST-standardized equivalents.

GEC exists to fill a particular point in the design space of communication security for resource constrained machines. This space is not so small that GEC could possibly be optimal for all users. For example, certain algorithms might be preferred over for reasons of performance or standardization. Similarly, devices with more bandwidth might wish to obtain higher assurance by using a larger authentication tag. This document is an explanation of the design and implementation of GEC - it should not be viewed as an argument against other design decisions.