The Veil Cryptosystem

Coda Hale

September 6, 2022

Veil is a public-key cryptosystem that provides confidentiality, authenticity, and integrity services for messages of arbitrary sizes and multiple receivers. This document describes its cryptographic constructions, their security properties, and how they are combined to implement Veil's feature set.

Contents

List of Algorithms

1 Motivation

Veil is a clean-slate effort to build a secure, asynchronous, PGP-like messaging cryptosystem using modern tools and techniques to resist new attacks from modern adversaries. PGP provides confidential and authentic multi-receiver messaging, but with many deficiencies.

1.1 Cryptographic Agility

PGP was initially released in 1991 using a symmetric algorithm called BassOmatic invented by Phil Zimmerman himself. Since then, it's supported IDEA, DES, Triple-DES, CAST5, Blowfish, SAFER-SK128, AES-128, AES-192, AES-256, Twofish, and Camellia, with proposed support for ChaCha20. For hash algorithms, it's supported MD5, SHA-1, RIPE-MD160, MD2, "double-width SHA", TIGER/192, HAVAL, SHA2-256, SHA2-384, SHA2-512, SHA2-224. For public key encryption, it's supported RSA, ElGamal, Diffie-Hellman, and ECDH, all with different parameters. For digital signatures, it's supported RSA, DSA, ElGamal, ECDSA, and EdDSA, again, all with different parameters.

As Langley [26] said regarding TLS:

> Cryptographic agility is a huge cost. Implementing and supporting multiple algorithms means more code. More code begets more bugs. More things in general means less academic focus on any one thing, and less testing and code-review per thing. Any increase in the number of options also means more combinations and a higher chance for a bad interaction to arise.

At best, each of these algorithms represents a geometrically increasing burden on implementors, analysts, and users. At worst, they represent a catastrophic risk to the security of the system [28, 15].

A modern system would use a limited number of cryptographic primitives and use a single instance of each.

1.2 Informal Constructions

PGP messages use a Sign-Then-Encrypt ($\mathcal{S}t\mathcal{E}$) construction, which is insecure given an encryption oracle [2, p. 41]:

> In the $\mathcal{S}t\mathcal{E}$ scheme, the adversary $\mathcal{A}$ can easily break the sUF-CMA security in the outsider model. It can ask [the encryption oracle] to signcrypt a message m for R' and get $C = (\textsc{Encrypt}(pk_{R'}, m||\sigma), ID_S, ID_{R'})$, where $\sigma \xleftarrow{R} \textsc{Sign}(pk_S, m)$. Then, it can recover $m||\sigma$ using $sk_{R'}$ and forge the signcryption ciphertext $C' = (\textsc{Encrypt}(pk_R, m||\sigma), ID_S, ID_R)$.

This may seem like an academic distinction, but this attack is trivial to mount. If you send your boss an angry resignation letter signed and encrypted with PGP, your boss can re-transmit that to your future boss, encrypted with her public key.

A modern system would use established, analyzed constructions with proofs in established models to achieve established notions with reasonable reductions to weak assumptions.

1.3 Non-Repudiation

A standard property of digital signatures is that of *non-repudiation*, or the inability of the signer to deny they signed a message. Any possessor of the signer's public key, a message, and a signature can verify the signature for themselves. For explicitly signed, public messages, this is a very desirable property. For encrypted, confidential messages, this is not.

Similar to the vindictive boss scenario above, an encrypted-then-signed PGP message can be decrypted by an intended receiver (or someone in possession of their private key) and presented to a third party as an unencrypted, signed message without having to reveal anything about themselves. The inability of PGP to preserve the privacy context of confidential messages should rightfully have a chilling effect on its users [16].

A modern system would be designed to provide some level of deniability to confidential messages.

1.4 Global Passive Adversaries

A new type of adversary which became immediately relevant to the post-Snowden era is the Global Passive Adversary, which monitors all traffic on all links of a network. For an adversary with an advantaged network position (e.g. a totalitarian state), looking for cryptographically-protected messages is trivial given the metadata they often expose. Even privacy features like GnuPG's `--hidden-recipients` still produce encrypted messages which are trivially identifiable as encrypted messages, because PGP messages consist of packets with explicitly identifiable metadata. In addition to being secure, privacy-enhancing technologies must be undetectable.

Bernstein, Hamburg, Krasnova, and Lange [9] summarized this dilemma:

> Cryptography hides patterns in user data but does not evade censorship if the censor can recognize patterns in the cryptography itself.

A modern system would produce messages without recognizable metadata or patterns.

2 Security Model And Notions

2.1 Multi-User Confidentiality

To evaluate the confidentiality of a scheme, we consider an adversary $\mathcal{A}$ attempting to attack a sender and receiver [5, p. 44]. $\mathcal{A}$ creates two equal-length messages (m_0, m_1), the sender selects one at random and encrypts it, and $\mathcal{A}$ guesses which of the two has been encrypted without tricking the receiver into decrypting it for them. To model real-world possibilities, we assume $\mathcal{A}$ has three capabilities:

1. $\mathcal{A}$ can create their own key pairs. Veil does not have a centralized certificate authority and creating new key pairs is intentionally trivial.
2. $\mathcal{A}$ can trick the sender into encrypting arbitrary plaintexts with arbitrary public keys. This allows us to model real-world flaws such as servers which return encrypted error messages with client-provided data [36].
3. $\mathcal{A}$ can trick the receiver into decrypting arbitrary ciphertexts from arbitrary senders. This allows us to model real-world flaws such as padding oracles [33].

Given these capabilities, $\mathcal{A}$ can mount an attack in two different settings: the outsider setting and the insider setting.

2.1.1 Outsider Confidentiality

In the multi-user outsider model, we assume $\mathcal{A}$ knows the public keys of all users but none of their private keys [5, p. 44].

The multi-user outsider model is useful in evaluating the strength of a scheme against adversaries who have access to some aspect of the sender and receiver's interaction with messages (e.g. a padding oracle) but who have not compromised the private keys of either.

2.1.2 Insider Confidentiality

In the multi-user insider model, we assume $\mathcal{A}$ knows the sender's private key in addition to the public keys of both users [5, p. 45–46].

The multi-user insider model is useful in evaluating the strength of a scheme against adversaries who have compromised a user.

Forward Sender Security A scheme which provides confidentiality in the multi-user insider setting is called *forward sender secure* because an adversary who compromises a sender cannot read messages that sender has previously encrypted [19].

2.2 Multi-User Authenticity

To evaluate the authenticity of a scheme, we consider an adversary $\mathcal{A}$ attempting to attack a sender and receiver [5, p. 47]. $\mathcal{A}$ attempts to forge a ciphertext which the receiver will decrypt but which the sender never encrypted. To model real-world possibilities, we again assume $\mathcal{A}$ has three capabilities:

1. $\mathcal{A}$ can create their own key pairs.
2. $\mathcal{A}$ can trick the sender into encrypting arbitrary plaintexts with arbitrary public keys.
3. $\mathcal{A}$ can trick the receiver into decrypting arbitrary ciphertexts from arbitrary senders.

As with multi-user confidentiality, this can happen in the outsider setting and the insider setting.

2.2.1 Outsider Authenticity

In the multi-user outsider model, we again assume $\mathcal{A}$ knows the public keys of all users but none of their private keys [5, p. 47].

Again, this is useful to evaluate the strength of a scheme in which $\mathcal{A}$ has some insight into senders and receivers but has not compromised either.

2.2.2 Insider Authenticity

In the multi-user insider model, we assume $\mathcal{A}$ knows the receiver's private key in addition to the public keys of both users [5, p. 48].

Key Compromise Impersonation A scheme which provides authenticity in the multi-user insider setting effectively resists *key compromise impersonation*, in which $\mathcal{A}$, given knowledge of a receiver's private key, can forge messages to that receiver from arbitrary senders [35]. The classic example is authenticated Diffie-Hellman (e.g. RFC 9180 [6, 1]), in which the static Diffie-Hellman shared secret point $K = [d_S]Q_R$ is used to encrypt a message and its equivalent $K' = [d_R]Q_S$ is used to decrypt it. An attacker in possession of the receiver's private key d_R and the sender's public key Q_S can simply encrypt the message using $K' = [d_R]Q_S$ without ever having knowledge of d_S. Digital signatures are a critical element of schemes which provide insider authenticity, as they give receivers a way to verify the authenticity of a message using authenticators they (or an adversary with their private key) could never construct themselves.

2.3 Insider vs. Outsider Security

The multi-receiver setting motivates a focus on insider security over the traditional emphasis on outsider security (contra [2, p. 26][5, p. 46]; see [4]). Given a probability of an individual key compromise P, a multi-user system of N users has an overall

$1 - (1 - P)^N$ probability of at least one key being compromised. A system with an exponentially increasing likelihood of losing all confidentiality and authenticity properties is not acceptable.

2.4 Indistinguishable From Random Noise

Indistinguishability from random noise is a critical property for censorship-resistant communication [9]:

> Censorship-circumvention tools are in an arms race against censors. The censors study all traffic passing into and out of their controlled sphere, and try to disable censorship-circumvention tools without completely shutting down the Internet. Tools aim to shape their traffic patterns to match unblocked programs, so that simple traffic profiling cannot identify the tools within a reasonable number of traces; the censors respond by deploying firewalls with increasingly sophisticated deep-packet inspection.
>
> Cryptography hides patterns in user data but does not evade censorship if the censor can recognize patterns in the cryptography itself.

2.5 Limited Deniability

The inability of a receiver (or an adversary in possession of a receiver's private key) to prove the authenticity of a message to a third party is critical for privacy. Other privacy-sensitive protocols achieve this by forfeiting insider authenticity or authenticity altogether [16]. Veil achieves a limited version of deniability: a receiver can only prove the authenticity of a message to a third party by revealing their own private key. This deters a dishonest receiver from selectively leaking messages and requires all-or-nothing disclosure from an adversary who compromises an honest receiver.

3 Cryptographic Primitives

In the interests of cryptographic minimalism, Veil uses just two distinct cryptographic primitives:

- Keccyak-128 [21, 11] for confidentiality, authentication, and integrity.
- jq255e for key agreement and authenticity [31].

3.1 Keccyak-128

Keccyak-128 is the adaptation of the Cyclist duplex construction from Xoodyak [21] to the Keccak-p[1600,12] permutation instead of Xoodoo (thus "Keccyak").

Cyclist is a permutation-based cryptographic duplex, a cryptographic primitive that provides symmetric-key confidentiality, integrity, and authentication via a single object [21]. Duplexes offer a way to replace complex, ad-hoc constructions combining encryption algorithms, cipher modes, AEADs, MACs, and hash algorithms using a single primitive [21, 10]. Duplexes have security properties which reduce to the properties of the cryptographic sponge, which themselves reduce to the strength of the underlying permutation [13].

The Keccak-p[1600,12] permutation is the basis of the KangarooTwelve hash algorithm [11] and allows for much higher throughput in software than the Xoodoo permutation at the expense of requiring a larger state. It has a 1600-bit width, like Keccak-f[1600] (the basis of SHA-3), but with a reduced number of rounds for speed. Xoodyak's Cyclist parameters of $R_{\text{hash}} = b - 256$, $R_{\text{kin}} = b - 32$, $R_{\text{kout}} = b - 192$, and $\ell_{\text{ratchet}} = 16$ are adapted for the larger permutation width of $b = 1600$. The resulting construction targets a 128-bit security level and is ~4x faster than SHA-256, ~5x faster than ChaCha20Poly1305, and ~7.5x faster than AES-128-GCM in software.

Veil's security assumes that Cyclist's Encrypt operation is IND-CPA secure, its Squeeze operation is sUF-CMA secure, and its Encrypt/Squeeze-based authenticated encryption construction is IND-CCA2 secure.

3.2 jq255e

jq255e is a double-odd elliptic curve selected for efficiency and simplicity [30, 31]. It provides a prime-order group, has non-malleable encodings, and has no co-factor concerns. This allows for the use of a wide variety of cryptographic constructions built on group operations. It targets a 128-bit security level, lends itself to constant-time implementations, and can run in constrained environments.

Veil's security assumes that the Gap Discrete Logarithm and Gap Diffie-Hellman problems are hard relative to jq255e.

4 Construction Techniques

Veil uses a few common construction techniques in its design which bear specific mention.

4.1 Unkeyed And Keyed Duplexes

Veil uses Cyclist, which offers both unkeyed ("hash") and keyed modes. All Veil constructions begin in the unkeyed mode by absorbing a constant domain separation string (e.g. `veil.mres`). To convert from an unkeyed duplex to a keyed duplex, a 512-bit key is derived from the unkeyed duplex's state and used to initialize a keyed duplex:

Algorithm 1 Converting an unkeyed Cyclist duplex to a keyed duplex.

```
Absorb(example.domain)                ▷ Initialize an unkeyed duplex.
Absorb($X$)                           ▷ Absorb some input data.
$K \leftarrow$ SqueezeKey(64)         ▷ Squeeze a 512-bit key.
Cyclist($K$, $\epsilon$, $\epsilon$)  ▷ Initialize a keyed duplex.
$C \leftarrow$ Encrypt($P$)           ▷ Use the keyed duplex.
```

The unkeyed duplex is used as a kind of key derivation function, with the lower absorb rate of Cyclist's unkeyed mode providing better avalanching properties.

4.2 Integrated Constructions

Cyclist is a cryptographic duplex, thus each operation is cryptographically dependent on the previous operations. Veil makes use of this by integrating different types of constructions to produce a single, unified construction. Instead of having to pass forward specific values (e.g. hashes of values or derived keys) to ensure cryptographic dependency, Cyclist allows for constructions which simply absorb all values, thus ensuring transcript integrity of complex protocols.

For example, a traditional hybrid encryption scheme like HPKE [6] will describe a key encapsulation mechanism (KEM) like X25519 and a data encapsulation mechanism (DEM) like AES-GCM and link the two together via a key derivation function (KDF) like HKDF by deriving a key and nonce for the DEM from the KEM output.

In contrast, the same construction using Cyclist would be the following three operations, in order (2):

Algorithm 2 HPKE in Cyclist.

```
1: Cyclist($[d_E]Q_R$, $\epsilon$, $\epsilon$)
2: $C \leftarrow$ Encrypt($P$)
3: $T \leftarrow$ Squeeze(16)
```

The duplex is keyed with the shared secret point (line 1), used to encrypt the plaintext (line 2), and finally used to squeeze an authentication tag (line 3). Each operation modifies the duplex's state, making the final Squeeze operation dependent on both the previous Encrypt operation (and its argument, P) but also the Cyclist operation before it.

This is both a dramatically clearer way of expressing the overall hybrid public-key encryption construction and more efficient: because the ephemeral shared secret point is unique, no nonce need be derived (or no all-zero nonce need be justified in an audit).

4.2.1 Process History As Hidden State

A subtle but critical benefit of integrating constructions via a cryptographic duplex is that authenticators produced via Squeeze operations are dependent on the entire process history of the duplex, not just on the emitted ciphertext. The DEM components of Alg. 2 (i.e. Encrypt/Squeeze) are superficially similar to an Encrypt-then-MAC ($\mathcal{E}t\mathcal{M}$) construction, but where an adversary in possession of the MAC key can forge authenticators given an $\mathcal{E}t\mathcal{M}$ ciphertext, the duplex-based approach makes that infeasible. The output of the Squeeze operation is dependent not just on the keying material (i.e. the Cyclist operation) but also on the plaintext P. An adversary attempting to forge an authenticator given only key material and ciphertext will be unable to reconstruct the duplex's state and thus unable to compute their forgery.

4.3 Hedged Ephemeral Values

When generating ephemeral values, Veil uses Aranha et al.'s "hedged signature" technique [3] to mitigate against both catastrophic randomness failures and differential fault attacks against purely deterministic schemes.

Specifically, the duplex's state is cloned, and the clone absorbs a context-specific secret value (e.g. the signer's private key in a digital signature scheme) and a 64-byte random value. The clone duplex is used to produce the ephemeral value or values for the scheme.

For example, the following operations would be performed on the cloned duplex (3):

Algorithm 3 Hedged ephemeral generation with Cyclist.

```
with clone do                                   ▷ Clone the duplex's state.
    Absorb($d$)                                 ▷ Absorb a private key.
    $v \xleftarrow{\$} \{0,1\}^{512}$           ▷ Generate a random value.
    Absorb($v$)                                 ▷ Absorb the random value.
    $x \leftarrow$ Squeeze(32) mod $\ell$       ▷ Squeeze a hedged ephemeral scalar.
    yield $x$                                   ▷ Return $x$ to the outer context.
end                                             ▷ Destroy the cloned duplex's state.
```

The ephemeral scalar x is returned to the context of the original construction and the cloned duplex is discarded. This ensures that even in the event of a catastrophic failure of the random number generator, x is still unique relative to d. Depending on the uniqueness needs of the construction, an ephemeral value can be hedged with a plaintext in addition to a private key.

5 Digital Signatures

`veil.schnorr` implements a Schnorr digital signature scheme.

5.1 Signing A Message

Signing a message is described in full in Alg. 4.

Algorithm 4 Signing a message M with a key pair (d, Q).

```
function SIGN($(d_S, Q_S), M$)
    ABSORB(veil.schnorr)                    ▷ Initialize an unkeyed duplex.
    ABSORB($Q$)                             ▷ Absorb the signer's public key.
    ABSORB($M$)                             ▷ Absorb the message.

    with clone do                           ▷ Clone the duplex's state.
        ABSORB($d$)                         ▷ Absorb the sender's private key.
        $v \xleftarrow{\$} \{0,1\}^{512}$   ▷ Generate a random value.
        ABSORB($v$)                         ▷ Absorb the random value.
        $k \leftarrow$ SQUEEZE(32) mod $\ell$
        yield $k$                           ▷ Yield a hedged commitment scalar.
    end

    CYCLIST(SQUEEZEKEY(64), $\epsilon$, $\epsilon$)   ▷ Convert to a keyed duplex.

    $I \leftarrow [k]G$                     ▷ Calculate the commitment point.
    $S_0 \leftarrow$ ENCRYPT($I$)           ▷ Encrypt the commitment point.

    $r \leftarrow$ SQUEEZE(16) mod $\ell$   ▷ Squeeze a short challenge scalar.
    $s \leftarrow dr + k$                   ▷ Calculate the proof scalar.
    $S_1 \leftarrow$ ENCRYPT($s$)           ▷ Encrypt the proof scalar.

    return $S = S_0 || S_1$
end function
```

5.2 Verifying A Signature

Verifying a signature is described in full in Alg. 5.

5.3 Constructive Analysis Of `veil.schnorr`

The Schnorr signature scheme is the application of the Fiat-Shamir transform to the Schnorr identification scheme.

Algorithm 5 Verifying a signature S with a message M and a public key Q.

```
function Verify($Q, M, S = S_0||S_1$)
    Absorb(veil.schnorr)                    ▷ Initialize an unkeyed duplex.
    Absorb($Q$)                             ▷ Absorb the signer's public key.
    Absorb($M$)                             ▷ Absorb the message.

    Cyclist(SqueezeKey(64), $\epsilon$, $\epsilon$)           ▷ Convert to a keyed duplex.

    $I \leftarrow$ Decrypt($S_0$)                   ▷ Decrypt the commitment point.
    $r' \leftarrow$ Squeeze(16) mod $\ell$            ▷ Squeeze a short challenge scalar.

    $s \leftarrow$ Decrypt($S_1$)                   ▷ Decrypt the proof scalar.
    $I' \leftarrow [s]G - [r']Q$                ▷ Calculate the counterfactual commitment point.

    return $I' \overset{?}{=} I$                  ▷ The signature is valid if both points are equal.
end function
```

Unlike Construction 13.12 of Katz and Lindell [24, p. 482], `veil.schnorr` transmits the commitment point I as part of the signature and the verifier calculates I' vs transmitting the challenge scalar r and calculating r'. In this way, `veil.schnorr` is closer to EdDSA [17] or the Schnorr variant proposed by Hamburg [22]. Short challenge scalars are used which allow for faster verification with no loss in security [31]. In addition, this construction allows for the use of variable-time optimizations during signature verification [32].

5.4 UF-CMA Security

Per Theorem 13.10 of Katz and Lindell [24, p. 478], this construction is UF-CMA secure if the Schnorr identification scheme is secure and the hash function is secure:

> Let Π be an identification scheme, and let Π' be the signature scheme that results by applying the Fiat-Shamir transform to it. If Π is secure and H is modeled as a random oracle, then Π' is secure.

Per Theorem 13.11 of Katz and Lindell [24, p. 481], the security of the Schnorr identification scheme is conditioned on the hardness of the discrete logarithm problem:

> If the discrete-logarithm problem is hard relative to $\mathcal{G}$, then the Schnorr identification scheme is secure.

Per Sec 5.10 of Bertoni, Daemen, Peeters, and Van Assche [12], Cyclist is a suitable random oracle if the underlying permutation is indistinguishable from a random permutation. Thus, `veil.schnorr` is UF-CMA if the discrete-logarithm problem is hard relative to jq255e and Keccak-p is indistinguishable from a random permutation.

5.5 sUF-CMA Security

Some Schnorr/EdDSA implementations (e.g. Ed25519) suffer from malleability issues, allowing for multiple valid signatures for a given signer and message [17]. Chalkias, Garillot, and Nikolaenko [20] describe a strict verification function for Ed25519 which achieves sUF-CMA security in addition to strong binding:

1. Reject the signature if $S \notin \{0, \ldots, L-1\}$.
2. Reject the signature if the public key A is one of 8 small order points.
3. Reject the signature if A or R are non-canonical.
4. Compute the hash $\text{SHA2}_{512}(R||A||M)$ and reduce it mod L to get a scalar h.
5. Accept if $8(S \cdot B) - 8R - 8(h \cdot A) = 0$.

Rejecting $S \geq L$ makes the scheme sUF-CMA secure, and rejecting small order A values makes the scheme strongly binding. `veil.schnorr`'s use of canonical point and scalar encoding routines obviate the need for these checks. Likewise, jq255e is a prime order group, which obviates the need for cofactoring in verification.

When implemented with a prime order group and canonical encoding routines, the Schnorr signature scheme is strongly unforgeable under chosen message attack (sUF-CMA) in the random oracle model and even with practical cryptographic hash functions [29, 27].

5.6 Key Privacy

The EdDSA variant (i.e. $S = (I, s)$) is used over the traditional Schnorr construction (i.e. $S = (r, s)$) to enable the variable-time computation of $I' = [s]G - [r]Q$, which provides a ~30% performance improvement. That construction, however, allows for the recovery of the signing public key given a signature and a message: given the commitment point I, one can calculate $Q = -[r^{-1}](I - [s]G)$.

For Veil, this behavior is not desirable. A global passive adversary should not be able to discover the identity of a signer from a signed message.

To eliminate this possibility, `veil.schnorr` encrypts both components of the signature with a duplex keyed with the signer's public key in addition to the message. An attack which recovers the plaintext of either signature component in the absence of the public key would imply that Cyclist is not IND-CPA.

5.7 Indistinguishability From Random Noise

Given that both signature components are encrypted with Cyclist, an attack which distinguishes between a `veil.schnorr` and random noise would also imply that Cyclist is not IND-CPA.

6 Encrypted Headers

veil.sres implements a single-receiver, deniable signcryption scheme which Veil uses to encrypt message headers. It integrates an ephemeral ECDH KEM, a Cyclist DEM, and a designated-verifier Schnorr signature scheme to provide multi-user insider security.

6.1 Encrypting A Header

Encrypting a header is described in full in Alg. 6.

Algorithm 6 Encrypting a header with sender's key pair (d_S, Q_S), ephemeral key pair (d_E, Q_E), receiver's public key Q_R, nonce N, and plaintext P.

```
function EncryptHeader($(d_S, Q_S), (d_E, Q_E), Q_R, N, P$)
    Absorb(veil.sres)                              ▷ Initialize an unkeyed duplex.
    Absorb($Q_S$)                                  ▷ Absorb the sender's public key.
    Absorb($Q_R$)                                  ▷ Absorb the receiver's public key.
    Absorb($N$)                                    ▷ Absorb the nonce.
    Absorb($d_S[Q_R]$)                             ▷ Absorb the static ECDH shared secret.
    Cyclist(SqueezeKey(64), $\epsilon$, $\epsilon$)  ▷ Convert to a keyed duplex.

    $C_0 \leftarrow$ Encrypt($Q_E$)                ▷ Encrypt the ephemeral public key.
    Absorb($d_E[Q_R]$)                             ▷ Absorb the ephemeral ECDH shared secret.
    $C_1 \leftarrow$ Encrypt($P$)                  ▷ Encrypt the plaintext.

    with clone do                                  ▷ Clone the duplex's state.
        Absorb($d_S$)                              ▷ Absorb the sender's private key.
        $v \xleftarrow{\$} \{0,1\}^{512}$          ▷ Generate a random value.
        Absorb($v$)                                ▷ Absorb the random value.
        $k \leftarrow$ Squeeze(32) mod $\ell$      ▷ Squeeze a commitment scalar.
        yield $k$
    end

    $I \leftarrow [k]G$                            ▷ Calculate the commitment point.
    $S_0 \leftarrow$ Encrypt($I$)                  ▷ Encrypt the commitment point.

    $r \leftarrow$ Squeeze(32) mod $\ell$          ▷ Squeeze a challenge scalar.
    $s \leftarrow d_S r + k$                       ▷ Calculate the proof scalar.

    $X \leftarrow [s]Q_R$                          ▷ Calculate the proof point.
    $S_1 \leftarrow$ Encrypt($X$)                  ▷ Encrypt the proof point.

    return $C_0 || C_1 || S_0 || S_1$
end function
```

6.2 Decrypting A Header

Decrypting a header is described in full in Alg. 7.

Algorithm 7 Decrypting a header with receiver's key pair (d_R, Q_R), sender's public key Q_S, nonce N, and ciphertext $C_0||C_1||S_0||S_1$.

```
function DecryptHeader($(d_R, Q_R), Q_S, N, C_0||C_1||S_0||S_1$)
    Absorb(veil.sres)                              ▷ Initialize an unkeyed duplex.
    Absorb($Q_S$)                                  ▷ Absorb the sender's public key.
    Absorb($Q_R$)                                  ▷ Absorb the receiver's public key.
    Absorb($N$)                                    ▷ Absorb the nonce.

    Absorb($d_R[Q_S]$)                             ▷ Absorb the static ECDH shared secret.
    Cyclist(SqueezeKey(64), $\epsilon$, $\epsilon$)            ▷ Convert to a keyed duplex.

    ${Q_E}' \leftarrow$ Decrypt($C_0$)                     ▷ Decrypt the ephemeral public key.
    Absorb($d_R[{Q_E}']$)                          ▷ Absorb the ephemeral ECDH shared secret.
    $P' \leftarrow$ Decrypt($C_1$)                        ▷ Decrypt the plaintext.

    $I \leftarrow$ Decrypt($S_0$)                         ▷ Decrypt the commitment point.
    $r' \leftarrow$ Squeeze(32) mod $\ell$                  ▷ Squeeze a challenge scalar.

    $X \leftarrow$ Decrypt($S_1$)                         ▷ Decrypt the proof point.
    $X' \leftarrow [d_R](I + [r']Q_S)$                   ▷ Re-calculate the proof point.

    if $X' \stackrel{?}{=} X$ then                          ▷ Ensure the ciphertext is authentic.
        return ${Q_E}', P'$
    else
        return $\perp$
    end if
end function
```

6.3 Constructive Analysis Of `veil.sres`

`veil.sres` is an integration of two well-known constructions: an ECIES-style hybrid public key encryption scheme and a designated-verifier Schnorr signature scheme.

The initial portion of `veil.sres` is equivalent to ECIES (see Construction 12.23 of [24, p. 435]), (with the commitment point I as an addition to the ciphertext, and the challenge scalar r serving as the authentication tag for the data encapsulation mechanism) and is IND-CCA2 secure (see Corollary 12.14 of [24, p. 436]).

The latter portion of `veil.sres` is a designated-verifier Schnorr signature scheme which adapts an EdDSA-style Schnorr signature scheme by multiplying the proof scalar s by the receiver's public key Q_R to produce a designated-verifier point X [34]. The

EdDSA-style Schnorr signature is sUF-CMA secure when implemented in a prime order group and a cryptographic hash function [17, 20, 29, 27] (see also Sec. 5).

6.4 Multi-User Confidentiality

One of the two main goals of the `veil.sres` is confidentiality in the multi-user setting (see Sec. 2.1), or the inability of an adversary $\mathcal{A}$ to learn information about plaintexts.

6.4.1 Outsider Confidentiality

First, we evaluate the confidentiality of `veil.sres` in the multi-user outsider setting (see Sec. 2.1.1), in which the adversary $\mathcal{A}$ knows the public keys of all users but none of their private keys [5, p. 44].

The classic multi-user attack on the generic Encrypt-Then-Sign ($\mathcal{E}t\mathcal{S}$) construction sees $\mathcal{A}$ strip the signature σ from the challenge ciphertext

$$C = (c, \sigma, Q_S, Q_R)$$

and replace it with

$$\sigma' \xleftarrow{\$} \text{Sign}(d_{\mathcal{A}}, c)$$

to produce an attacker ciphertext

$$C' = (c, \sigma', Q_{\mathcal{A}}, Q_R)$$

at which point $\mathcal{A}$ can trick the receiver into decrypting the result and giving $\mathcal{A}$ to the randomly-chosen plaintext $m_0 \vee m_1$ [2, p. 40]. This attack is not possible with `veil.sres`, as the sender's public key is strongly bound during encryption (see Alg. 6, Line 3) and decryption (see Alg. 7, Line 3).

$\mathcal{A}$ is unable to forge valid signatures for existing ciphertexts, limiting them to passive attacks. A passive attack on any of the three components of `veil.sres` ciphertexts–C, S_0, S_1–would only be possible if Cyclist is not IND-CPA secure.

Therefore, `veil.sres` provides confidentiality in the multi-user outsider setting.

6.4.2 Insider Confidentiality

Next, we evaluate the confidentiality of `veil.sres` in the multi-user insider setting (see Sec. 2.1.2), in which the adversary $\mathcal{A}$ knows the sender's private key in addition to the public keys of both users [5, p. 45–46].

$\mathcal{A}$ cannot decrypt the message by themselves, as they do not know either d_E or d_R and cannot calculate the ECDH shared secret $[d_E]Q_R = [d_R]Q_E = [d_E d_R G]$.

$\mathcal{A}$ also cannot trick the receiver into decrypting an equivalent message by replacing the signature, despite $\mathcal{A}$'s ability to use d_S to create new signatures. In order to generate a valid signature on a ciphertext c' (e.g. $c' = c||1$), $\mathcal{A}$ would have to squeeze a valid challenge scalar r' from the duplex state (see Alg. 6, Line 24). Unlike the signature hash function in the generic $\mathcal{E}t\mathcal{S}$ composition, however, the duplex state is cryptographically dependent on values $\mathcal{A}$ does not know, specifically the ECDH shared secret $[d_E]Q_S$ (via the ABSORB operation) and the plaintext P (via the ENCRYPT operation).

Therefore, `veil.sres` provides confidentiality in the multi-user insider setting.

6.5 Multi-User Authenticity

The second of the two main goals of the `veil.sres` is authenticity in the multi-user setting (see Sec. 2.2), or the inability of an adversary $\mathcal{A}$ to forge valid ciphertexts.

6.5.1 Outsider Authenticity

First, we evaluate the authenticity of `veil.sres` in the multi-user outsider setting (see Sec. 2.2.1), in which the adversary $\mathcal{A}$ knows the public keys of all users but none of their private keys [5, p. 47].

Because the Schnorr signature scheme is sUF-CMA secure, it is infeasible for $\mathcal{A}$ to forge a signature for a new message or modify an existing signature for an existing message. Therefore, `veil.sres` provides authenticity in the multi-user outsider setting.

6.5.2 Insider Authenticity

Next, we evaluate the authenticity of `veil.sres` in the multi-user insider setting (see Sec. 2.2.2), in which the adversary $\mathcal{A}$ knows the receiver's private key in addition to the public keys of both users [5, p. 48].

Again, the Schnorr signature scheme is sUF-CMA secure and the signature is created using the signer's private key. The receiver (or $\mathcal{A}$ in possession of the receiver's private key) cannot forge signatures for new messages. Therefore, `veil.sres` provides authenticity in the multi-user insider setting.

6.6 Limited Deniability

`veil.sres`'s use of a designated-verifier Schnorr scheme provides limited deniability for senders (see Sec. 2.5). Without revealing d_R, the receiver cannot prove the authenticity of a message (including the identity of its sender) to a third party.

6.7 Indistinguishability From Random Noise

All of the components of a `veil.sres` ciphertext–C, S_0, and S_1–are Cyclist ciphertexts. An adversary in the outsider setting (i.e. knowing only public keys) is unable to calculate any of the key material used to produce the ciphertexts; a distinguishing attack is infeasible if Cyclist is IND-CPA secure.

6.8 Re-use Of Ephemeral Keys

The re-use of an ephemeral key pair (d_E, Q_E) across multiple ciphertexts does not impair the confidentiality of the scheme provided (N, Q_R) pairs are not re-used [8]. An adversary who compromises a retained ephemeral private key would be able to decrypt all messages the sender encrypted using that ephemeral key, thus the forward sender security is bounded by the sender's retention of the ephemeral private key.

7 Encrypted Messages

`veil.mres` implements a multi-receiver signcryption scheme.

7.1 Encrypting A Message

Encrypting a message is described in full in Alg. 8.

7.2 Decrypting A Message

Decrypting a signature is described in full in Alg. 9.

7.3 Constructive Analysis Of `veil.mres`

`veil.mres` is an integration of two well-known constructions: a multi-receiver hybrid encryption scheme and an EdDSA-style Schnorr signature scheme.

The initial portion of `veil.mres` is a multi-receiver hybrid encryption scheme, with per-receiver copies of a symmetric data encryption key (DEK) encrypted in headers with the receivers' public keys [25, 8, 7, 18]. The headers are encrypted with the `veil.sres` construction (see Sec. 6), which provides full insider security (i.e. IND-CCA2 and sUF-CMA in the multi-user insider setting), using a per-header Squeeze value as a nonce. The message itself is divided into a sequence of 32KiB blocks, each encrypted with a sequence of Cyclist Encrypt/Squeeze operations, which is IND-CCA2 secure.

The latter portion of `veil.mres` is an EdDSA-style Schnorr signature scheme. The EdDSA-style Schnorr signature is sUF-CMA secure when implemented in a prime order group and a cryptographic hash function [17, 20, 29, 27] (see also Sec. 5). Short challenge scalars are used which allow for faster verification with no loss in security [31]. In addition, this construction allows for the use of variable-time optimizations during signature verification [32].

7.4 Multi-User Confidentiality

One of the two main goals of the `veil.mres` is confidentiality in the multi-user setting (see Sec. 2.1), or the inability of an adversary $\mathcal{A}$ to learn information about plaintexts. As `veil.mres` is a multi-receiver scheme, we adopt Bellare et al.'s adaptation of the multi-user setting, in which $\mathcal{A}$ may compromise a subset of receivers [7].

7.4.1 Outsider Confidentiality

First, we evaluate the confidentiality of `veil.mres` in the multi-user outsider setting (see Sec. 2.1.1), in which the adversary $\mathcal{A}$ knows the public keys of all users but none of their private keys [5, p. 44].

As with `veil.sres` (see Sec. 6.4), `veil.mres` superficially resembles an Encrypt-Then-Sign ($\mathcal{E}t\mathcal{S}$) scheme, which are vulnerable to an attack where by $\mathcal{A}$ strips the signature from the challenge ciphertext and either signs it themselves or tricks the sender into signing it,

Algorithm 8 Encrypting a message with sender's key pair (d_S, Q_S), receiver public keys $Q_{R^0}..Q_{R^n}$, padding length N_P, and plaintext P.

```
function EncryptMessage($(d_S, Q_S), Q_{R^0}..Q_{R^n}, N_P, P$)
    Absorb(veil.mres)                                  ▷ Initialize an unkeyed duplex.
    Absorb($Q_S$)                                      ▷ Absorb the sender's public key.

    with clone do                                      ▷ Clone the duplex's state.
        Absorb($d_S$)                                  ▷ Absorb the sender's private key.
        $v \xleftarrow{\$} \{0,1\}^{512}$              ▷ Generate a random value.
        Absorb($v$)                                    ▷ Absorb the random value.
        $k \leftarrow$ Squeeze(32) mod $\ell$          ▷ Squeeze a commitment scalar.
        $d_E \leftarrow$ Squeeze(32) mod $\ell$        ▷ Squeeze an ephemeral private key.
        $Q_E = [d_E]G$                                 ▷ Calculate the ephemeral public key.
        $K \leftarrow$ Squeeze(32)                     ▷ Squeeze a data encryption key.
        $N \leftarrow$ Squeeze(16)                     ▷ Squeeze a nonce.
        yield $(k, d_E, Q_E, K)$
    end

    $C \leftarrow N$                                   ▷ Write the nonce.
    Absorb($N$)                                        ▷ Absorb the nonce.

    $H = K||N_Q||N_P$                                  ▷ Encode the DEK and params in a header.
    for all $Q_{R^i} \in \{Q_{R^0}..Q_{R^n}\}$ do      ▷ Encrypt the header for each receiver.
        $N \leftarrow$Squeeze(16)
        $H \leftarrow$EncryptHeader($(d_S, Q_S), (d_E, Q_E), Q_{R^i}, H, N$)
        Absorb($H$)
        $C \leftarrow C||H$
    end for

    $y \xleftarrow{\$} \{0,1\}^{N_P}$                  ▷ Generate random padding.
    Absorb($y$)                                        ▷ Absorb padding.
    $C \leftarrow C||y$                                ▷ Append padding to ciphertext.

    Absorb($K$)                                        ▷ Absorb the DEK.
    Cyclist(SqueezeKey(64), $\epsilon$, $\epsilon$)    ▷ Convert to a keyed duplex.

    for all 32-KiB blocks $p \in P$ do
        $C \leftarrow C||$Encrypt($p$)                 ▷ Encrypt and tag each block.
        $C \leftarrow C||$Squeeze(16)
    end for

    $I \leftarrow [k]G$                                ▷ Calculate and encrypt the commitment point.
    $C \leftarrow C||$Encrypt($I$)

    $r \leftarrow$ Squeeze(16) mod $\ell$              ▷ Squeeze a short challenge scalar.
    $s \leftarrow d_S r + k$                           ▷ Calculate and encrypt the proof scalar.
    $C \leftarrow C||$Encrypt($s$)
    return $C$
end function
```

Algorithm 9 Decrypting a message with receiver's key pair (d_R, Q_R), sender's public key Q_S, and ciphertext C.

```
function DecryptMessage($(d_R, Q_R), Q_S, C$)
    Absorb(veil.mres)                                    ▷ Initialize an unkeyed duplex.
    Absorb($Q_S$)                                        ▷ Absorb the sender's public key.
    Absorb($C[0..16]$)                                   ▷ Absorb the nonce.

    for all possible encrypted headers $h \in C$ do
        $N \leftarrow$Squeeze(16)
        $x \leftarrow$ DecryptHeader($(d_R, Q_R), Q_S, h, N$)
        if $x \neq \perp$ then
            $Q_E, K||N_Q||N_P \leftarrow x$
            break
        end if
        Absorb($h$)
    end for

    Absorb($C[16 + N_Q..16 + N_Q + N_P]$)                ▷ Absorb padding.
    $C \leftarrow C[16 + N_Q + N_P..]$                   ▷ Skip to the message beginning.

    Absorb($K$)                                          ▷ Absorb the DEK.
    Cyclist(SqueezeKey(64), $\epsilon$, $\epsilon$)      ▷ Convert to a keyed duplex.

    $P' = \epsilon$                                      ▷ Initialize a buffer for the plaintext.
    for all 32KiB+16 blocks $c||t \in C$ do
        $P' \leftarrow P'||$Decrypt($c$)
        if Squeeze(16) $\neq t$ then                     ▷ Check each block's tag.
            return $\perp$
        end if
    end for

    $S_0||S_1 \leftarrow C$
    $I \leftarrow$ Decrypt($S_0$)                        ▷ Decrypt the commitment point.
    $r' \leftarrow$ Squeeze(16) mod $\ell$               ▷ Squeeze a short challenge scalar.

    $s \leftarrow$ Decrypt($S_1$)                        ▷ Decrypt the proof scalar.
    $I' \leftarrow [s]G - [r']Q_E$                       ▷ Calculate the counterfactual commitment point.

    if $I' \stackrel{?}{=} I$ then                       ▷ Verify the signature.
        return $P'$                                      ▷ Return the plaintext.
    else
        return $\perp$
    end if
end function
```

thereby creating a new ciphertext they can then trick the receiver into decrypting for them. Again, as with `veil.sres`, the identity of the sender is strongly bound during encryption encryption (see Alg. 8, Line 3) and decryption (see Alg. 9, Line 3), making this infeasible.

$\mathcal{A}$ is unable to forge valid signatures for existing ciphertexts, limiting them to passive attacks. `veil.mres` ciphertexts consist of ephemeral keys, encrypted headers, random padding, encrypted message blocks, and encrypted signature points. Each component of the ciphertext is dependent on the previous inputs (including the headers, which use Squeeze-derived nonce to link the `veil.sres` ciphertexts to the `veil.mres` state). A passive attack on any of those would only be possible if Cyclist is not IND-CPA secure.

7.4.2 Insider Confidentiality

Next, we evaluate the confidentiality of `veil.mres` in the multi-user insider setting (see Sec. 2.1.2), in which the adversary $\mathcal{A}$ knows the sender's private key in addition to the public keys of all users [5, p. 45–46]. $\mathcal{A}$ cannot decrypt the message by themselves, as they do not know either d_E or any d_R and cannot decrypt any of the `veil.sres`-encrypted headers. As with `veil.sres` (see Sec. 6.4), $\mathcal{A}$ cannot trick the receiver into decrypting an equivalent message by replacing the signature, despite $\mathcal{A}$'s ability to use d_S to create new headers. In order to generate a valid signature on a ciphertext c' (e.g. $c' = c||1$), $\mathcal{A}$ would have to squeeze a valid challenge scalar r' from the duplex state (see Alg. 8, Line 43). Unlike the signature hash function in the generic $\mathcal{E}t\mathcal{S}$ composition, however, the duplex state is cryptographically dependent on a value $\mathcal{A}$ does not know, specifically the data encryption key K (via the Absorb operation) and the plaintext blocks $p_{0..n}$ (via the Encrypt operation).

Therefore, `veil.mres` provides confidentiality in the multi-user insider setting.

7.5 Multi-User Authenticity

The second of the two main goals of the `veil.mres` is authenticity in the multi-user setting (see Sec. 2.2), or the inability of an adversary $\mathcal{A}$ to forge valid ciphertexts.

7.5.1 Outsider Authenticity

First, we evaluate the authenticity of `veil.mres` in the multi-user outsider setting (see Sec. 2.2.1), in which the adversary $\mathcal{A}$ knows the public keys of all users but none of their private keys [5, p. 47].

Because the Schnorr signature scheme is sUF-CMA secure, it is infeasible for $\mathcal{A}$ to forge a signature for a new message or modify an existing signature for an existing message. Therefore, `veil.mres` provides authenticity in the multi-user outsider setting.

7.5.2 Insider Authenticity

Next, we evaluate the authenticity of `veil.mres` in the multi-user insider setting (see Sec. 2.2.2), in which the adversary $\mathcal{A}$ knows some receivers' private keys in addition to the public keys of both users [5, p. 48].

Again, the Schnorr signature scheme is sUF-CMA secure and the signature is created using the ephemeral private key, which $\mathcal{A}$ does not possess. The receiver (or $\mathcal{A}$ in possession of the receiver's private key) cannot forge signatures for new messages. Therefore, `veil.mres` provides authenticity in the multi-user insider setting.

7.6 Limited Deniability

The only portion of `veil.mres` ciphertexts which are creating using the sender's private key (and thus tying a particular message to their identity) are the `veil.sres`-encrypted headers. All other components are creating using the data encryption key or ephemeral private key, neither of which are bound to identity. `veil.sres` provides limited deniability (see Sec. 6.6), therefore `veil.mres` does as well.

7.7 Indistinguishability From Random Noise

`veil.mres` ciphertexts are indistinguishable from random noise. All components of an `veil.mres` ciphertext are Cyclist ciphertexts; a successful distinguishing attack on them would require Cyclist to not be IND-CPA secure.

7.8 Partial Decryption

The division of the plaintext stream into blocks takes its inspiration from the CHAIN construction [23], but the use of Cyclist allows for a significant reduction in complexity. Instead of using the nonce and associated data to create a feed-forward ciphertext dependency, the Cyclist duplex ensures all encryption operations are cryptographically dependent on the ciphertext of all previous encryption operations. Likewise, because the `veil.mres` ciphertext is terminated with a Schnorr signature (see Sec. 5), using a special operation for the final message block isn't required.

The major limitation of such a system is the possibility of the partial decryption of invalid ciphertexts. If an attacker flips a bit on the fourth block of a ciphertext, `veil.mres` will successfully decrypt the first three before returning an error. If the end-user interface displays that, the attacker may be successful in radically altering the semantics of an encrypted message without the user's awareness. The first three blocks of a message, for example, could say `PAY MALLORY $100`, `GIVE HER YOUR CAR`, `DO WHAT SHE SAYS`, while the last block might read `JUST KIDDING`.

8 Passphrase-Based Encryption

`veil.pbenc` implements a memory-hard authenticated encryption scheme to encrypt private keys at rest.

8.1 Encrypting A Private Key

Encrypting a private key is described in full in Alg. 10.

Algorithm 10 Encrypting a private key d given a passphrase P, memory cost parameter C_M, time cost parameter C_T, and parallelism cost parameter C_P.

```
function EncryptPrivateKey($P, M_{cost}, T_{cost}, d$)
    Absorb(veil.pbenc)                               ▷ Initialize an unkeyed duplex.
    $P_N \leftarrow$ NormalizeNFC($P$)                        ▷ Normalize the passphrase.
    $S \xleftarrow{\$} \{0,1\}^{128}$                              ▷ Generate a random salt.
    $K \leftarrow$ Argon2id(0x13, $P_N, S, C_M, C_T, C_P, 64$)        ▷ Hash the passphrase.
    Absorb($K$)                                       ▷ Absorb the derived key.
    Cyclist(SqueezeKey(64), $\epsilon$, $\epsilon$)               ▷ Convert to a keyed duplex.
    $C \leftarrow$ Encrypt($d$)                              ▷ Encrypt the private key.
    $T \leftarrow$ Squeeze(16)                        ▷ Squeeze an authentication tag.
    return $C_M||C_T||C_P||S||C||T$
end function
```

8.2 Decrypting A Private key

Decrypting a private key is described in full in Alg. 11.

8.3 Constructive Analysis Of `veil.pbenc`

`veil.pbenc` integrates Argon2 [14], a memory-hard key derivation function, and a standard Cyclist authenticated encryption scheme. Argon2 increases the cost for an attacker to mount a brute-force or dictionary attack using highly parallel hardware (e.g. GPUs, ASICs) by increasing the per-core memory requirements for attempts.

Algorithm 11 Decrypting a private key given a passphrase P and ciphertext C.

```
function DecryptPrivateKey($P, C = C_M || C_T || C_P || S || C || T$)
    Absorb(veil.pbenc)                              ▷ Initialize an unkeyed duplex.
    $P_N$ ←NormalizeNFC($P$)                          ▷ Normalize the passphrase.
    $S \xleftarrow{\$} \{0,1\}^{128}$                 ▷ Generate a random salt.
    $K$ ←Argon2id(0x13, $P_N, S, C_M, C_T, C_P, 64$)  ▷ Hash the passphrase.
    Absorb($K$)                                     ▷ Absorb the derived key.
    Cyclist(SqueezeKey(64), $\epsilon$, $\epsilon$)  ▷ Convert to a keyed duplex.
    $d'$ ←Decrypt($C$)                                ▷ Decrypt the ciphertext.
    $T'$ ←Squeeze(16)                                 ▷ Squeeze a counterfactual tag.
    if $T' \overset{?}{=} T$ then                     ▷ Authenticate the ciphertext.
        return $d'$
    else
        return $\perp$
    end if
end function
```

9 Message Digests & Authentication Codes

Veil can create message digests given a sequence of metadata values and a message.

9.1 Creating A Message Digest

Creating a message digest is described in full in Alg. 12.

Algorithm 12 Creating a message digest with metadata strings V and message M.

```
function DIGEST($V, M$)
    ABSORB(veil.digest)                          ▷ Initialize an unkeyed duplex.
    for all $V_i \in V$ do
        ABSORB($V_i$)                            ▷ Absorb metadata strings in order.
    end for
    ABSORB($M$)                                  ▷ Absorb message.
    $D \leftarrow$SQUEEZE(32)                    ▷ Squeeze a 32-byte digest.
    return $D$
end function
```

9.2 Creating A Message Authentication Code

By passing a symmetric key as a metadata string, `veil.digest` can be adapted to produce message authentication codes (see Alg. 13).

Algorithm 13 Creating a message authentication code with key K, metadata strings V, and message M.

```
function MAC($K, V, M$)
    $T \leftarrow$DIGEST($K||V,\ M$)             ▷ Include the key as metadata.
    return $T$
end function
```

9.3 Preimage Security and Collision Resistance

The duplex is indistinguishable from a random oracle given that the underlying permutation is indistinguishable from a random permutation [12, Sec. 8.5]. If Keccak-p is strong, an attacker has negligible advantage in finding first or second preimages.

9.4 sUF-CMA Security

Similarly, the security of a duplex as a MAC reduces to the strength of the permutation [12, Sec. 8.6]. An attack which forges a message for a given key and MAC would imply Cyclist's SQUEEZE operation is not sUF-CMA secure.

10 References

References

[1] J. Alwen, B. Blanchet, E. Hauck, E. Kiltz, B. Lipp, and D. Riepel. Analysing the HPKE standard. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 87–116. Springer, 2021. URL: https://eprint.iacr.org/2020/1499.pdf.

[2] J. H. An and T. Rabin. Security for signcryption: the two-user model. In *Practical Signcryption*, pages 21–42. Springer, 2010.

[3] D. F. Aranha, C. Orlandi, A. Takahashi, and G. Zaverucha. Security of hedged Fiat–Shamir signatures under fault attacks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 644–674. Springer, 2020. URL: https://eprint.iacr.org/2019/956.pdf.

[4] C. Badertscher, F. Banfi, and U. Maurer. A constructive perspective on signcryption security. In *IACR Cryptol. ePrint Arch.* 2018. URL: https://ia.cr/2018/050.

[5] J. Baek and R. Steinfeld. Security for signcryption: the multi-user model. In *Practical Signcryption*, pages 43–53. Springer, 2010.

[6] R. Barnes, K. Bhargavan, B. Lipp, and C. Wood. Hybrid public key encryption. Feb. 2022. URL: http://www.rfc-editor.org/rfc/rfc9180.html.

[7] M. Bellare, A. Boldyreva, K. Kurosawa, and J. Staddon. Multi-recipient encryption schemes: efficient constructions and their security. *IEEE Transactions on Information Theory*, 53(11):3927–3943, 2007. URL: https://faculty.cc.gatech.edu/~aboldyre/papers/bbks.pdf.

[8] M. Bellare, A. Boldyreva, and J. Staddon. Randomness re-use in multi-recipient encryption schemes. In *International Workshop on Public Key Cryptography*, pages 85–99. Springer, 2003. URL: https://www.iacr.org/archive/pkc2003/25670085/25670085.pdf.

[9] D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 967–980, 2013. URL: https://elligator.cr.yp.to/elligator-20130828.pdf.

[10] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. Duplexing the sponge: single-pass authenticated encryption and other applications. In *International Workshop on Selected Areas in Cryptography*, pages 320–337. Springer, 2011. URL: https://keccak.team/files/SpongeDuplex.pdf.

[11] G. Bertoni, J. Daemen, M. Peeters, G. V. Assche, R. V. Keer, and B. Viguier. Kangarootwelve: fast hashing based on Keccak-p. In *International Conference on Applied Cryptography and Network Security*, pages 400–418. Springer, 2018. URL: https://eprint.iacr.org/2016/770.pdf.

[12] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Cryptographic sponge functions. In *SHA-3 competition (round 3)*, 2011. URL: https://keccak.team/files/CSF-0.1.pdf.

[13] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. On the indifferentiability of the sponge construction. In N. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, pages 181–197, Berlin, Heidelberg. Springer Berlin Heidelberg, 2008. ISBN: 978-3-540-78967-3. URL: https://keccak.team/files/SpongeIndifferentiability.pdf.

[14] A. Biryukov, D. Dinu, D. Khovratovich, and S. Josefsson. Argon2 memory-hard function for password hashing and proof-of-work applications. Aug. 2021. URL: http://www.rfc-editor.org/rfc/rfc9106.html.

[15] J. Blessing, M. A. Specter, and D. J. Weitzner. You really shouldn't roll your own crypto: an empirical study of vulnerabilities in cryptographic libraries. *CoRR*, abs/2107.04940, 2021. arXiv: 2107.04940. URL: https://arxiv.org/abs/2107.04940.

[16] N. Borisov, I. Goldberg, and E. Brewer. Off-the-record communication, or, why not to use PGP. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 77–84, 2004. URL: https://otr.cypherpunks.ca/otr-wpes.pdf.

[17] J. Brendel, C. Cremers, D. Jackson, and M. Zhao. The provable security of Ed25519: theory and practice. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1659–1676, 2021. DOI: 10.1109/SP40001.2021.00042. URL: https://eprint.iacr.org/2020/823.pdf.

[18] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. OpenPGP message format. Nov. 2007. URL: http://www.rfc-editor.org/rfc/rfc4880.html.

[19] R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 255–271. Springer, 2003. URL: https://eprint.iacr.org/2003/083.pdf.

[20] K. Chalkias, F. Garillot, and V. Nikolaenko. Taming the many EdDSAs. In *International Conference on Research in Security Standardisation*, pages 67–90. Springer, 2020. URL: https://eprint.iacr.org/2020/1244.pdf.

[21] J. Daemen, S. Hoffert, M. Peeters, G. Van Assche, and R. Van Keer. Xoodyak, a lightweight cryptographic scheme. *IACR Transactions on Symmetric Cryptology*, 2020(S1):60–87, June 2020. DOI: 10.13154/tosc.v2020.iS1.60-87. URL: https://tosc.iacr.org/index.php/ToSC/article/view/8618.

[22] M. Hamburg. The STROBE protocol framework. 2017. URL: https://eprint.iacr.org/2017/003.pdf.

[23] V. T. Hoang, R. Reyhanitabar, P. Rogaway, and D. Vizár. Online authenticated-encryption and its nonce-reuse misuse-resistance. In *Annual Cryptology Conference*, pages 493–517. Springer, 2015. URL: https://eprint.iacr.org/2015/189.pdf.

[24] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC, Dec. 2020. DOI: 10.1201/9781351133036. URL: https://doi.org/10.1201/9781351133036.

[25] K. Kurosawa. Multi-recipient public-key encryption with shortened ciphertext. In *International Workshop on Public Key Cryptography*, pages 48–63. Springer, 2002. URL: https://eprint.iacr.org/2001/071.

[26] A. Langley. Cryptographic agility. 2016. URL: https://www.imperialviolet.org/2016/05/16/agility.html.

[27] G. Neven, N. P. Smart, and B. Warinschi. Hash function requirements for Schnorr signatures. *Journal of Mathematical Cryptology*, 3(1):69–87, 2009. URL: http://www.neven.org/papers/schnorr.pdf.

[28] P. Q. Nguyen. Can we trust cryptographic software? cryptographic flaws in GNU Privacy Guard v1.2.3. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 555–570. Springer, 2004. URL: https://link.springer.com/content/pdf/10.1007%2F978-3-540-24676-3_33.pdf.

[29] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000. URL: https://www.di.ens.fr/david.pointcheval/Documents/Papers/2000_joc.pdf.

[30] T. Pornin. Double-odd elliptic curves. 2020. URL: https://eprint.iacr.org/2020/1558.

[31] T. Pornin. Double-odd Jacobi quartic. 2022. URL: https://eprint.iacr.org/2022/1052.

[32] T. Pornin. Optimized lattice basis reduction in dimension 2, and fast Schnorr and EdDSA signature verification. 2020. URL: https://eprint.iacr.org/2020/454.

[33] J. Rizzo and T. Duong. Practical padding oracle attacks. In *4th USENIX Workshop on Offensive Technologies (WOOT 10)*, 2010. URL: https://www.usenix.org/legacy/event/woot10/tech/full_papers/Rizzo.pdf.

[34] R. Steinfeld, H. Wang, and J. Pieprzyk. Efficient extension of standard Schnorr/RSA signatures into universal designated-verifier signatures. In *International Workshop on Public Key Cryptography*, pages 86–100. Springer, 2004. URL: https://www.iacr.org/archive/pkc2004/29470087/29470087.pdf.

[35] M. A. Strangio. On the resilience of key agreement protocols to key compromise impersonation. In *European Public Key Infrastructure Workshop*, pages 233–247. Springer, 2006. URL: https://eprint.iacr.org/2006/252.pdf.

[36] T. Yu, S. Hartman, and K. Raeburn. The perils of unauthenticated encryption: Kerberos version 4. In *NDSS*, volume 4, pages 4–4, 2004. URL: https://web.mit.edu/tlyu/papers/krb4peril-ndss04.pdf.