Security Analysis for Randomness Improvements for Security Protocols

Liliya Akhmetzyanova
lah@cryptopro.ru

Cas Cremers
cremers@cispa.saarland

Luke Garratt
lgarratt@cisco.com

Stanislav V. Smyshlyaev
smyshsv@gmail.com

v1.0 – November 1st, 2018

Abstract

Many cryptographic mechanisms depend on the availability of secure random numbers. In practice, the sources of random numbers can be unreliable for many reasons. There exist ways to improve the reliability of randomness, but these often do not work well with practical constraints. One proposal to reduce the impact of untrusted randomness is the proposal by Cremers et al. [3], which aims to be effective in existing deployments.

In this document, we provide a security analysis of the construction in [3] (Revision 3) and elaborate on design choices and practical interpretations.

1 Introduction

All key exchange protocols (e.g., SSL/TLS, SSH, IKE, etc.) depend on the generation of secure random numbers. At the core of all of these protocols is a Diffie–Hellman key exchange of the following basic form: Alice chooses random number x, computes g^x and sends it to Bob. Bob similarly chooses random number y, computes g^y and sends it to Alice. Of course, each particular protocol has other essential details such as certificates, signatures, key derivation, comparing of transcripts and so on, but at the heart of all these protocols is the shared secret g^{xy}. Therefore, it is essential that x and y are generated as securely as possible.

In many operating systems, raw entropy comes in the form of events such as mouse movements or keystrokes. As large amounts of raw entropy is difficult to accumulate, Alice and Bob do not generate truly random numbers x and y for their key exchanges. Instead, they use the primitive of a cryptographically secure pseudorandom number generator (CSPRNG), which takes as input a seed, continually harvests more raw entropy and produces arbitrary many pseudorandom numbers as required. Since CSPRNGs are used to provide the essential secret g^{xy}, they clearly need to be as secure as possible.

However, the current problem we face is that all CSPRNGs depend on raw entropy in some form or another. Therefore, if the underlying randomness is not good, the secret g^{xy} is not secure. Real-world examples of poor random number generation include:

- The Debian OpenSSL random number generator vulnerability [1, 7];
- Predictable random numbers in Android's Java OpenSSL [6] leading to theft of bitcoins;

- The Dual EC random number generator backdoor [2];
- Random number generator on hardware that degrades over time;
- Servers that are deployed in settings where good randomness generation cannot be guaranteed;
- Internet of Things (IoT) devices with good (factory) keys but no good randomness generation after deployment.

We propose a solution to this problem. Our solution is inspired by the so-called "NAXOS trick" for key exchange protocols. In contrast to the NAXOS trick, our design is more generic and applies outside of the AKE domain, and we follow a much more conservative approach that enables the re-use of existing infrastructure and improved graceful degradation if it turns out the hash function's output may reveal partial information from the output.

From the academic literature on authenticated key exchange protocols, the NAXOS protocol [5] does not just compute pseudorandom numbers x and y from raw entropy and send g^x and g^y, but instead sends $g^{H(x,\mathsf{sk}_{\hat{A}})}$ and $g^{H(y,\mathsf{sk}_{\hat{B}})}$ where H is a random oracle, and sk_A and sk_B are the long-term secret keys of Alice and Bob respectively. Intuitively we can see that securely in this case should now depend on the pairs $(x,\ \mathsf{sk}_{\hat{A}})$ and $(y,\ \mathsf{sk}_{\hat{B}})$ being secure, as opposed to just x and y being secure. What is more, a protocol that uses $g^{H(x,\mathsf{sk}_{\hat{A}})}$ can behave in exactly the same way as one that uses g^x. The only essential difference is that $H(x, \mathsf{sk}_{\hat{A}})$ is a safer secret than x.

In this note, we take an alternative approach for real-world protocols. In particular, often the long-term term sk is in trusted hardware so it is impossible to have direct access to it to compute $H(x, \mathsf{sk})$. For many HSM deployments, this prevents the application of the NAXOS trick. Moreover, a choice needs to be made as to what function implements the random oracle H. We will also need to know precisely what security guarantees our construction provides. We will answer these questions and provide proofs that are construction meet precisely definition security guarantees under standard cryptographic assumptions. We will show that our construction improves the generation of pseudorandom numbers and provides concrete security guarantees for a generic class of protocols including SSL/TLS, SSH, IKE, etc. at negligible cost to efficiency.

Our wrapper construction

We propose a "wrapper" function around existing pseudorandom number generators, in contexts that have access to a party's signing algorithm. Let y be the output of the pseudorandom number generator. As in the main document [3], we define the wrapper to be:

$$\mathsf{PRF}(\mathtt{KDF}(y, \mathtt{H}(\mathsf{Sig}(\mathsf{sk}, \mathtt{tag}_1))), \mathtt{tag}_2, n),$$

where $\mathtt{tag}_1$ and $\mathtt{tag}_2$ are public and are chosen from some fixed sets $\mathcal{T}_1$ and $\mathcal{T}_2$ correspondingly and where $\mathtt{PRF}$ and $\mathtt{KDF}$ and instantiated with HKDF-expand and HKDF-extract respectively. We assume that the size of $\mathcal{T}_1$ and $\mathcal{T}_2$ is negligible in the size of the security parameter. We encourage the reader to see the main document [3] for full details.

Overview

In Section 2 we recall the standard properties of a CSPRNG. In Section 3 we define the security properties we will need for our primitives in our wrapper construction. In Section 4 we define our security model for our wrapper construction. In Section 5 we give a security theorem and formal game hopping security proof it fulfils the security properties we claim. We make additional notes on the real-world security of our wrapper construction in Section 6.

2 Background on CSPRNGs

Forward secure cryptographically secure pseudorandom number generators (CSPRNG) are already used as primitives in TLS and other protocols. Let Rand be a forward secure CSPRNG. By this we mean Rand satisfies the following two properties:

Property 1. Indistinguishable from random (CSPRNG)

A family of deterministic polynomial time computable functions $Rand_k \colon \{0,1\}^k \to \{0,1\}^{p(k)}$ for some polynomial p is a CSPRNG, if it stretches the length of its input ($p(k) > k$ for any k) and if its output is computationally indistinguishable from true randomness, i.e. for any probabilistic polynomial time algorithm A, which outputs 1 or 0 as a distinguisher,

$$\left| \Pr_{x \leftarrow \{0,1\}^k}[A(Rand(x)) = 1] - \Pr_{r \leftarrow \{0,1\}^{p(k)}}[A(r) = 1] \right| < \mu(k)$$

for some negligible function μ.

All this is saying is that, no efficient algorithm (with knowledge of how Rand works) can distinguish the output of Rand from randomness, when the initial seed is unknown and uniformly random. Note that achieving this property is non-trivial: many pseudorandom number generators may achieve output that looks statistically random, but does not satisfy this property. For instance, the function that hashes the initial seed to produce x. Then hashes x. Then hashes again, and so on, could produce statistically random output, but an efficient adversary that knows how this algorithm works can clearly distinguish this from purely random behaviour after seeing the first two blocks of output (the second being the hash of the first).

Andrew Yao showed that this definition is equivalent to the next-bit test (below).

Property 2. Resistance to state compromise extensions (forward secure)

Another property we want is for the CSPRNG to be forward secure. A forward-secure CSPRNG with block length $t(k)$ is a $Rand_k \colon \{0,1\}^k \to \{0,1\}^k \times \{0,1\}^{t(k)}$, where the input string s_i with length k is the current state at period i, and the output (s_{i+1}, y_i) consists of the next state s_{i+1} and the pseudorandom output block y_i of period i, such that it withstands state compromise extensions in the following sense. If the initial state s_1 is chosen uniformly at random from $\{0,1\}^k$, then for any i, the sequence $(y_1, y_2, \ldots, y_i, s_{i+1})$ must be computationally indistinguishable from $(r_1, r_2, \ldots, r_i, s_{i+1})$, in which the r_i are chosen uniformly at random from $\{0,1\}^{t(k)}$.

The next-bit test (equivalent definition of CSPRNG)

By satisfying the next-bit test, we mean that given the first k bits of outputs from Rand sequence, there is no polynomial-time algorithm that can predict the $(k+1)$th bit with probability of success better than 50% (without knowing the seed). Andrew Yao proved in 1982 that a generator passing the next-bit test will pass all other polynomial-time statistical tests for randomness.

Let P be a polynomial, and $S = \{S_k\}$ be a collection of sets such that S_k contains (k)-bit long sequences. Moreover, let μ_k be the probability distribution of the strings in S_k.

Let $\mathcal{M}$ be a probabilistic Turing machine, working in polynomial time. Let $p_{k,i}^{\mathcal{M}}$ be the probability that $\mathcal{M}$ predicts the $(i+1)$st bit correctly, i.e.

$$p_{k,i}^{\mathcal{M}} = \mathcal{P}[M(s_1 \ldots s_i) = s_{i+1} | s \in S_k \text{ with probability } \mu_k(s)]$$

We say that collection $S = \{S_k\}$ passes the next-bit test if for all polynomial Q, for all but finitely many k, for all $0 < i < k$:

$$p_{k,i}^{\mathcal{M}} < \frac{1}{2} + \frac{1}{Q(k)}$$

3 Security definitions

Here we will precisely define the security properties we need of our primitives for our construction to be proven secure in our security model.

3.1 Pseudorandom function

A pseudorandom function is an algorithm PRF. This algorithm implements a deterministic function $z = \mathsf{PRF}(k, x, n)$, taking as input a key k and some bit string x, and returning a string $z \in \{0,1\}^n$. For brevity we will omit the parameter n assuming that the PRF algorithm returns the values of the maximum permitted size.

We define a security game between an adversary and challenger as follows.

1. The challenger uniformly randomly samples t secret keys $k_1, \ldots, k_t$ independently from each other.
2. The adversary is allowed to query the challenger with adaptively chosen values (i, x), $i \in [1, \ldots, t]$. The challenger replies with $\mathsf{PRF}(k_i, x)$.
3. Eventually the adversary queries a special symbol T to indicate the so-called test query with value (i, x) that was not queried before. At this point, the challenger computes $z_0 \coloneqq \mathsf{PRF}(k_i, x)$ and samples z_1 uniformly randomly. The challenger then flips a coin $b \leftarrow_{\$} \{0,1\}$ and responds with z_b. Note that after the test query the adversary is not allowed to query (i, x).
4. The adversary then outputs a guess b' for b and wins the game if $b = b'$ and loses otherwise.

The advantage of any probabilistic polynomial time adversary winning the above security game with polynomial number of keys should be negligible in the security parameter. In other words,

$$\left| \Pr(b = b') - \frac{1}{2} \right| \leq \epsilon_{\mathsf{PRF}}$$

3.2 Key derivation function

The security property we require of $\mathtt{KDF}$ is as follows. Intuitively, we require $\mathtt{KDF}(y, x)$ to be indistinguishable from uniform random if at least one of x or y in unknown. In other words we assume that the $\mathtt{KDF}(\cdot, x)$ and $\mathtt{KDF}(y, \cdot)$ functions, where x and y are chosen uniformly independently at random from the corresponding sets, are computationally indistinguishable from two "ideal" functions chosen uniformly independently at random from the sets of all functions with corresponding domains and ranges. Let $\epsilon_{\mathtt{KDF}}$ denote the probability that any probabilistic polynomial time adversary is able to distinguish between these distributions.

In our concrete construction, we instantiate $\mathtt{KDF}$ with HKDF-extract. The HKDF-Extract scheme is proved to be a secure extractor that takes as inputs a value a salt and long-term key. In [4], the scheme is not analyzed in the case where the long-term key is compromised but the salt is not. For HKDF-extract based on HMAC, we believe it is reasonable to expect the above security

guarantee. Certainly in many cryptographic security proofs such a KDF is merely modeled as a random oracle and in those cases our assumption trivially follows.

3.3 Hash function and signature scheme

A signature scheme is a triple $(\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf})$. KGen is a probabilistic algorithm which takes as input the security parameter 1^k and outputs a public signature verification key pk and secret signing key sk. Sig is a signing algorithm which generates a signature σ for message m using secret key sk. In the current paper we consider only deterministic Sig algorithms. Vf is a deterministic signature verification algorithm which, given input (pk, σ, m), outputs 1 if σ is a valid signature of m under key pk, and 0 otherwise. It is required that for every k, every $(\mathsf{sk}, \mathsf{pk})$ output by $\mathsf{KGen}(1^k)$, and every message m, it holds that

$$\mathsf{Vf}(m, \mathsf{Sig}(\mathsf{sk}, m)) = 1$$

In our construction, we will not want to use the long-term key sk directly. We will instead use the hash of a signature of $\mathtt{tag}_1$, signed with sk. For our security proof, we require the combined hash function and signature scheme to fulfil the following security property.

Consider the following game between a challenger and a polynomial time adversary:

1. The challenger generates a public/private key pair $(\mathsf{pk}, \mathsf{sk})$ using KGen and gives the adversary the public key pk. Then the adversary sends the $\mathtt{tag}_1$ value to the challenger.
2. The adversary is allowed to query adaptively chosen messages $m_1, \ldots, m_q$ for some $q \in \mathbb{N}$ to the challenger so long as none of the queries m_i are $\mathtt{tag}_1$. The challenger responds to each query with $\sigma_i = \mathsf{Sig}(\mathsf{sk}, m_i)$.
3. When the adversary decides to, it outputs a so-called test query. At this point, the challenger flips an unbiased coin $b \leftarrow_{\$} \{0, 1\}$. If heads, it returns with $\mathtt{H}(\mathsf{Sig}(\mathsf{sk}, \mathtt{tag}_1))$. If tails, it responds with a uniformly random string of the same length.
4. The adversary is allowed to keep asking for signatures, but eventually it must output a guess b' for the coin flip b, at which point the game ends. The adversary wins if it guesses the coin flip correctly and loses otherwise.

We require that the advantage of any probabilistic polynomial time adversary winning the above game is negligible in the security parameter. In other words,

$$\left| \Pr(b = b') - \frac{1}{2} \right| \leq \epsilon_{\mathsf{Sig},\mathtt{H}}$$

This security property can be instantiated with routine cryptographic assumptions such as a hash function behaving as a random oracle and an existentially unforgeable signature scheme (where $\mathtt{tag}_1$ is not allowed to be queried).

4 Security model

Here we define the security property we expect from our wrapper in terms of a game between a challenger and an adversary.

We run a game between the challenger and the adversary as follows. At the beginning of the game the challenger uniformly randomly chooses a secret key sk and returns a public key pk

to the adversary. Receiving the public key the adversary chooses a set $T \subseteq \mathcal{T}_1$ consisting of l pairwise different values $t_1, \ldots, t_l$ and sends it to the challenger. We assume that the maximum size n of the PRF output and the size l of the set T are polynomial.

The adversary is allowed to make the following queries to the challenger.

- The adversary can make `output` queries of two types:
  - $\mathtt{tag}_1, \mathtt{tag}_2$ (in this case the challenger chooses the y value uniformly randomly by itself);
  - $\mathtt{tag}_1, \mathtt{tag}_2, y$ (in this case the y value is chosen by the adversary).

  Note that the adversary is allowed to make queries where $\mathtt{tag}_1 \in T$.

  The challenger produces

  $$\mathsf{PRF}(\mathtt{KDF}(y, \mathtt{H}(\mathsf{Sig}(\mathsf{sk}, \mathtt{tag}_1)), \mathtt{tag}_2)$$

  and returns this value as a response to the corresponding `output` query.

  The challenger indexes queries and locally saves the corresponding inputs to the wrapper for each query, i.e. the challenger saves records of the form $(i, \mathtt{tag}_1^i, \mathtt{tag}_2^i, y_i)$. We assume that $i \leq M$ for some M that is also polynomial.

- The adversary can make `sign` queries for signatures with sk of messages m. The challenger produces and returns the value $\mathsf{Sig}(\mathsf{sk}, m)$.

- The adversary can make `corrupt` query for sk at any time. The challenger must respond with sk to this query.

- The adversary is also allowed to make `reveal` queries for the y_i value used in the ith `output` query at any time (the adversary makes the query with the index of the target `output` query). The challenger must respond with the y_i value used in generating the response to the ith `output` query.

- We say that the ith `output` query is *fresh* if one of the values y_i and `signature` stay secret. That is, if one of the following conditions is satisfied:
  - the adversary has not made the `corrupt` query or the `sign` query for $m = \mathtt{tag}_1^i$.
  - this query contains only $\mathtt{tag}_1, \mathtt{tag}_2$ and the adversary has not made the `reveal` query for y_i.

  We also say that an `output` query is "non-trivial" if one of the following conditions is satisfied

  - this query is of the type $\mathtt{tag}_1, \mathtt{tag}_2, y$ and this query has not been made before.
  - this query is of the type $\mathtt{tag}_1, \mathtt{tag}_2$.

  At some point in time, the adversary must make a so-called test `output` query. This is the same as a normal non-trivial `output` query except the challenger flips an unbiased coin and either responds with the genuine output using the wrapper, or a uniformly randomly chosen string of the same length. The adversary is allowed to query for the y used in the test if it wants to, as well as to continue making other queries. The adversary may also query for the sk value or for the signature of $\mathtt{tag}_1$ if it has not already done so. However, the test output query must remain fresh at all times during the game.

- At some point in time after the test query, the adversary must output a single bit as a guess. The adversary wins the game if it is able to guess the coin flip with non-negligible advantage over $\frac{1}{2}$.

5 Proof of security

Theorem 5.1. *If* PRF*,* KDF*,* Sig *and* $\mathtt{H}$ *satisfy the security definitions above, then any probabilistic polynomial time adversary has only negligible advantage in winning the security game.*

Proof. Let Game 0 denote the original security game as defined in our security model definition. Let S_i denote the event of the adversary winning Game i. Our goal in this proof is to bound $\Pr(S_0)$ to show that it is only at most negligibly above $\frac{1}{2}$. Although the security argument is very intuitive ("the adversary must surely need both y and sk to guess the secret") we will formally prove it in a game hopping proof.

At some point in time, the adversary must issue a `test` query. Let $\mathtt{tag}_1^i, \mathtt{tag}_2^i, y_i$ denote the values used by the challenger for this query. We prove this theorem in a case partition on whether the adversary has revealed $\mathsf{Sig}(\mathsf{sk}, \mathtt{tag}_1^i)$ or y_i. (If the adversary has queried for neither, then it is clearly in an even worse position to win the game.)

Let Game 1 be identical to Game 0 except the challenger guesses in advance an integer $i^\star \in [1, \ldots, M]$ and aborts (and the adversary loses) unless $i^\star = i$. In other words, the challenger needs to guess in advance which of the possible `output` queries will be the test. This is a large failure event game hop and it is easy to see that

$$\Pr(S_0) \leq M \Pr(S_1)$$

We now proceed with our case partition.

Case 1: The adversary has revealed y_i or has chosen y_i by itself.

In this case, the adversary is not allowed to query for sk or for the signature of $\mathtt{tag}_1^i$, otherwise the test output query would not be fresh.

Game 2. Let Game 2 be identical to Game 1 except the challenger guesses in advance an integer $j \in [1, \ldots, l]$ and aborts (and the adversary loses) unless $\mathtt{tag}_1^i = t_j$. This is a large failure event game hop and it is easy to see that

$$\Pr(S_1) \leq l \Pr(S_2)$$

Game 3. Define Game 3 to be identical to Game 2 except $\mathtt{H}(\mathsf{Sig}(\mathsf{sk}, \mathtt{tag}_1^i))$ is replaced with a value x_i sampled uniformly at random for each `output` query with $\mathtt{tag}_1 = \mathtt{tag}_1^i$. Here we claim that

$$\Pr(S_2) \leq \Pr(S_3) + \epsilon_{\mathtt{H},\mathsf{Sig}}$$

In particular, no probabilistic polynomial time distinguisher algorithm can distinguish between Game 2 and Game 3, since this would imply a way to beat the hash-signature game with better than $\epsilon_{\mathtt{H},\mathsf{Sig}}$ advantage. Precisely, an adversary in the hash-signature game could beat it with better than $\epsilon_{\mathtt{H},\mathsf{Sig}}$ advantage as follows.

It acts as a challenger in the hybrid game and inserts the value of pk from the hash-signature game. As an adversary in the hash-signature game, it inserts the known $\mathtt{tag}_1$ value from the previous game and asks a test query straight away, receiving either $\mathtt{H}(\mathsf{Sig}(\mathsf{sk}, \mathtt{tag}_1^i))$ or a uniformly randomly chosen value as per the rules of the hash-signature game. It inserts this value as the value for $\mathtt{H}(\mathsf{Sig}(\mathsf{sk}, \mathtt{tag}_1^i))$ in the hybrid game. Because of Game 2, the adversary knows where to

make this swap in the game. The output queries with $\mathtt{tag}_1 \neq t_j$ are processed by the adversary using queries for signature to its challenger. The adversary can now simulate output and test queries as normal using this value. The adversary does not have to worry about simulating a corrupt query because we are in case 1. Finally, it can simulate signature queries by merely forwarding them along into the hash-signature game. This perfectly simulates the hybrid game: it is literally Game 2 if the test returns $\mathtt{H}(\mathsf{Sig}(\mathsf{sk}, \mathtt{tag}_1^i))$, and it is literally Game 3 if it is a uniformly randomly chosen value. The adversary follows the coin flip choice in the simulated hybrid game as its guess for the hash-signature game. By assumption of the hardness of the hash-signature game, all adversaries can only win with at most advantage $\epsilon_{\mathtt{H},\mathsf{Sig}}$. Therefore, the difference in advantages of adversaries across Game 2 and Game 3 can also only be separated by at most $\epsilon_{\mathtt{H},\mathsf{Sig}}$. Thus the claim above is proven.

Game 4. Define Game 4 to be identical to Game 3 except $\mathtt{KDF}(\cdot, x_i)$ is replaced with an ideal random function $\rho(\cdot)$ for each output query with $\mathtt{tag}_1 = t_j$. Here we claim that

$$\Pr(S_3) \leq \Pr(S_4) + \epsilon_{\mathtt{KDF}}$$

Here we present the simulation argument for the KDF game. Because of Game 2, the adversary knows where to make this swap in the game precisely for output queries with $\mathtt{tag}_1 = t_j$. The KDF adversary chooses to attack KDF for the second argument x and makes queries y to compute $\mathtt{KDF}(y, x_i)$ for required y. If the adversary has made the $(\mathtt{tag}_1, \mathtt{tag}_2)$ query, then the KDF adversary chooses y value uniformly randomly by itself and locally saves it. Note that it can answer the reveal query in this case. All other queries are simulated in the normal way.

Game 5. Define Game 5 to be identical to Game 4 except $\mathsf{PRF}(\rho(y_i), \mathtt{tag}_2^i)$ is replaced with a value chosen uniformly randomly for the test query.

Here we claim that

$$\Pr(S_4) \leq \Pr(S_5) + \epsilon_{\mathsf{PRF}}$$

Here we present the simulation argument for the PRF game with at most m keys. The output queries with $\mathtt{tag}_1 = t_j$ and different y are processed with the different keys by asking the PRF challenger with the suitable indexes in queries. Since the test query should be non-trivial, y_i or $\mathtt{tag}_2^i$ should be new (we neglect the probability that for the output query of the type $(\mathtt{tag}_1, \mathtt{tag}_2)$ the y_i value chosen by the challenger itself collides with the previous values). Therefore, the PRF adversary can use its test query to swap the $\mathsf{PRF}(\rho(y_i), \mathtt{tag}_2^i)$ with the value obtained as a response on the test query in the PRF game. All other queries are simulated in the normal way.

It is clearly impossible for the adversary to have any advantage in guessing the secret bit in Game 5 (in either case, the wrapper generates a uniformly randomly chosen string). As such, $\Pr(S_5) = \frac{1}{2}$. Thus in this case $\Pr(S_0) \leq \frac{1}{2} + Ml(\epsilon_{(\mathtt{H},\mathsf{Sig})} + \epsilon_{\mathtt{KDF}} + \epsilon_{\mathsf{PRF}})$ which is negligibly close to $\frac{1}{2}$.

Case 2: The adversary has revealed sk or $\mathsf{Sig}(\mathsf{sk}, \mathtt{tag}_1^i)$.

In this case, the adversary is not allowed to reveal y_i or to choose y_i by itself, otherwise the test output query would not be fresh. Thus, y_i is chosen uniformly randomly.

Game 2. This game is identical to Game 1 except $\mathtt{KDF}(y_i, \mathtt{H}(\mathsf{Sig}(\mathsf{sk}, \mathtt{tag}_1)))$ is replaced with a value k sampled uniformly at random.

We claim that

$$\Pr(S_1) \leq \Pr(S_2) + \epsilon_{\mathtt{KDF}}$$

In the simulation the adversary chooses to attack KDF for the first argument y and makes query $\mathtt{H}(\mathsf{Sig}(\mathsf{sk}, \mathsf{tag}_1^i))$ to compute $\mathtt{KDF}(y_i, \mathtt{H}(\mathsf{Sig}(\mathsf{sk}, \mathsf{tag}_1^i)))$. All other queries are simulated in the normal way.

Game 3. This is the same as Game 5 from the other case but here is enough to make queries only for one key in the PRF game. □

6 Real-world considerations

6.1 Design choices

Which of our primitives are chosen for real-world reasons and which are useful for the security proof? Our security proof could be done even more easily by using directly sk instead of $\mathsf{Sig}(\mathsf{sk}, \mathsf{tag}_1)$. However, in the real-world it is preferable to keep the secret key in an HSM and only perform certain operations on it such as making signatures. There is no guarantee that the signature is uniformly distributed as a key, so this is why we hash it and chose $\mathtt{HSig}(\mathsf{sk}, \mathsf{tag}_1)$ over sk. Consequently, this also is why one of the security assumptions of our theorem is the difficulty of the "hash-signature game". The practical interpretation we are asking of the challenger is merely that the signature is a secret the adversary cannot guess, just like sk.

The KDF and PRF constructions are useful in practice because they are easily available, and they help in the security proof. In fact, the security assumptions we require of them for the proof are very minimal indeed. The security argument is extremely trivial if they are instantiated as random oracles, as is common in many security proofs. In our concrete construction, we recommend to implement them as HKDF-extract and HKDF-expand respectively. As mentioned in Section 3.2, one should be aware of our assumption regarding the security of the HKDF scheme when the long-term key is compromised while the salt is not.

The tags are assumed to be known in our security model. In practice they may not be, which only adds an extra layer of protection in practice.

In no case can our wrapper construction be worse than not having it all. In the usual case, we merely depend on y_i being secret. With our wrapper construction, we essentially use $\mathtt{KDF}(y_i \| x)$ for an x which may or may not be known. One may wonder how the signature is generated: probabilistic or deterministic? Could this be a chicken and egg problem, noting that in many signature schemes, if random numbers are bad and the signature is leaked, then the secret key is leaked? Thus, if random numbers are bad in the first place, the signature will be bad, and thus the construction will not be of any use. This is why we require that the signature scheme is deterministic.

6.2 Practical interpretations

The security model and consequent proof refers heavily to the concept of "freshness", which intuitively encodes the adversary not winning the game trivially by making the obvious reveal queries. The practical interpretation of our security theorem is as follows. If the adversary learns only one of the signature or the usual randomness generated on one particular instance, then under the security assumptions on our primitives, the wrapper construction should output randomness that is indistinguishable from a random string.

Our security model also explicitly assumes that $\mathsf{Sig}(\mathsf{sk}, \mathsf{tag}_1)$ never appears externally elsewhere in the protocol so that the adversary has no hope of seeing it and using it. Otherwise, security degrades to the normal case of generating pseudorandom numbers. Of course, in practice actually forcing $\mathsf{Sig}(\mathsf{sk}, \mathsf{tag}_1)$ to appear may still be difficult. Note furthermore that these signatures are agent-specific, which improves containment. Thus, leaking or forging $\mathsf{Sig}(\mathsf{sk}_{\hat{A}}, \mathsf{tag}_1)$

for a specific agent $\hat{A}$ only affects $\hat{A}$, and has no consequences for other agents. The fact that in reality $\texttt{tag}_1$ is separate for each application and $\texttt{tag}_2$ changes also helps to prevent repeated random numbers when there is a poor entropy source.

References

[1] Paolo Abeni, Luciano Bello, and Maximiliano Bertacchini. Exploiting DSA-1571: How to break PFS in SSL with EDH, July 2008.

[2] Daniel J Bernstein, Tanja Lange, and Ruben Niederhagen. Dual EC: A Standardized Back Door. Technical report, Cryptology ePrint Archive, Report 2015/767, 2015.

[3] C. Cremers, L. Garratt, S. Smyshlyaev, N. Sullivan, and C. Wood. Randomness Improvements for Security Protocols – Revision 3, 2018. `https://www.ietf.org/id/draft-irtf-cfrg-randomness-improvements-03.txt`.

[4] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In *Annual Cryptology Conference*, pages 631–648. Springer, 2010.

[5] Brian LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In *Provable Security*, pages 1–16. Springer, 2007.

[6] Rob Marvin. Google admits an android crypto PRNG flaw led to Bitcoin heist (August 2013).

[7] Scott Yilek, Eric Rescorla, Hovav Shacham, Brandon Enright, and Stefan Savage. When private keys are public: results from the 2008 Debian OpenSSL vulnerability. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 15–27. ACM, 2009.