Randomness Improvements for Security Protocols

draft-irtf-cfrg-randomness-improvements

Cas Cremers (cremers@cispa.saarland)
Luke Garratt (lgarratt@cisco.com)
Stanislav Smyshlyaev (svs@cryptopro.ru)
Nick Sullivan (nick@cloudflare.com)
Christopher A. Wood (cawood@apple.com)

CFRG
IETF 103, October 2018, Bangkok

Brief overview

Motivation

Most security mechanisms completely depend on randomness quality.
But PRNGs can break or contain design flaws.

- Bugs: Debian bug, Android's JCA PRNG flaw.
- Backdoors: Dual_EC_DRBG.
- Any hardware RNG can degrade over time.
- Vulnerable joint system entropy pools.

$\Rightarrow$ it's better to have a safety net to avoid system compromise.

Brief overview

Rationale

- "NAXOS trick" (LaMacchia, Brian et al., "Stronger Security of Authenticated Key Exchange").
- Direct access to private keys is not always possible, no APIs.
- Reusing signature keys outside of intended scope is not a good practice in general. So a very careful analysis is needed.
- Provide a ready-to-use solution, not requiring further deep analysis.

$\Rightarrow$ provide a solution such that any call for entropy would better be improved in a described way.

The construction

Let G(·) — the output of some CSPRNG. When randomness is needed, instead of G(n) use

$$G'(n) = \text{Expand}(\text{Extract}(G(L), H(\text{Sig}(sk, tag1))), tag2, n),$$

Intermediate values (including G(L) and Sig(sk, tag1)) must be kept secret.

- tag1: Constant string bound to a specific device and protocol in use (e.g. a MAC address).
- tag2: Non-constant string that includes a timestamp or counter.

The virtual machines issue

- McGrew, Anderson, Fluhrer, Shenefeil, „PRNG Failures and TLS Vulnerabilities in the Wild“.
- Ristenpart, Yilek, „When Good Randomness Goes Bad: Virtual Machine Reset Vulnerabilities and Hedging Deployed Cryptography“.

To provide security in the cases of usage of CSPRNGs in virtual environments, it is RECOMMENDED to incorporate all available information specific to the process that would ensure the uniqueness of each tag1 value among different instances of VMs (including ones that were cloned or recovered from snapshots). [...]

The proposed construction cannot provide any guarantees of security if the CSPRNG state is cloned due to the VM snapshots or process forking. Thus tag1 SHOULD incorporate all available information about the environment, such as process attributes, VM user information, etc.

The virtual machines issue

- McGrew, Anderson, Fluhrer, Shenefeil, „PRNG Failures and TLS Vulnerabilities in the Wild“.
- Ristenpart, Yilek, „When Good Randomness Goes Bad: Virtual Machine Reset Vulnerabilities and Hedging Deployed Cryptography“.

To provide security in the cases of usage of CSPRNGs in virtual environments, it is RECOMMENDED to incorporate all available information specific to the process that would ensure the uniqueness of each tag1 value among different instances of VMs (including ones that were cloned or recovered from snapshots). [...]

The proposed construction cannot provide any guarantees of security if the CSPRNG state is cloned due to the VM snapshots or process forking. Thus tag1 SHOULD incorporate all available information about the environment, such as process attributes, VM user information, etc.

Additional clarifications

Weak initial entropy source as additional motivation

Initial entropy sources can also be weak or broken, and that would lead to insecurity of all CSPRNG instances seeded with them.

Usage with HSMs: what happens where

If a private key sk is stored and used inside an HSM, then the signature calculation is implemented inside it, while all other operations (including calculation of a hash function, Extract and Expand functions) can be implemented either inside or outside the HSM.

Additional recommendations

Precomputation can be kept

In systems where signature computations are expensive, G'(n) may be precomputed and pooled. This is possible since the construction depends solely upon the CSPRNG output and private key.

Requirements for tag1: desired secrecy of Sig(sk, tag1)

tag1 may have the format that is not supported (or explicitly forbidden) by other applications using sk.

Requirements for tag2

tag2 MUST be implemented such that its values never repeat. This means, in particular, that timestamp is guaranteed to change between two requests to CSPRNG (otherwise counters should be used).

Minor changes

A slightly different viewpoint for a whole document: defining the construction as a new CSPRNG G'.

Cleaning up use of „RNG“, „PRNG“, „CSPRNG“, „randomness“, „entropy“ etc.

Security proofs

Desired security properties

1. If the CSPRNG works fine, that is, in a certain adversary model the CSPRNG output is indistinguishable from a truly random sequence, then the output of the proposed construction is also indistinguishable from a truly random sequence in that adversary model.
2. An adversary Adv with full control of a (potentially broken) CSPRNG and able to observe all outputs of the proposed construction, does not obtain any non-negligible advantage in leaking the private key, modulo side channel attacks.
3. If the CSPRNG is broken or controlled by adversary Adv, the output of the proposed construction remains indistinguishable from random provided the private key remains unknown to Adv.

The paper

L. Akhmetzyanova, C. Cremers, L. Garratt, S. Smyshlyaev.
„Security Analysis for an Improved Randomness Wrapper“.
Cryptology ePrint Archive: Report 2018/1057,
https://eprint.iacr.org/2018/1057

Summary

- The property #2 (security of the private key) is trivial assuming that requirements for implementations (enumerated in the I-D) are met.
- Complete game-hopping proofs for (even stronger versions of) the properties #1 and #3, assuming rather basic properties of the used primitives.
- But something in the assumptions can still be improved.

The paper

L. Akhmetzyanova, C. Cremers, L. Garratt, S. Smyshlyaev.
„Security Analysis for an Improved Randomness Wrapper“.
Cryptology ePrint Archive: Report 2018/1057,
https://eprint.iacr.org/2018/1057

Summary

- The property #2 (security of the private key) is trivial assuming that requirements for implementations (enumerated in the I-D) are met.
- Complete game-hopping proofs for (even stronger versions of) the properties #1 and #3, assuming rather basic properties of the used primitives.
- But something in the assumptions can still be improved.

Adversary model

The adversary wants to:

- distinguish a certain output of the construction from random.

The adversary can

- choose tag1 and tag2 from the sets $\mathcal{T}_1$ and $\mathcal{T}_2$ for all queries;
- learn values generated by the inner CSPRNG or even select its values;
- select any output of the construction (not necessarily the final one) for attack;
- ask to reveal either the values generated by the inner CSPRNG or the private key sk (but not both) for attacked output

— we believe that the model perfectly reflects practice (and is much stronger than in practice in fact).

Assumptions

No problems in the random oracle model for KDF. But we can do better.

. . . with several additional assumptions:

- Extract(x, y) is indistinguishable from random function for known x and unknown y, and vice versa.
  - There is no proof of such a property for HKDF-extract(salt, IKM), though it's reasonable to expect such a property.
  - Will try to prove it or make minor changes to the construction.
- Intermediate values of Sig(sk, tag1) are kept secret during the computations (I-D requires that implementations do this).
- sk is never used to sign values from $\mathcal{T}_1$ outside of the construction, limits on $\mathcal{T}_1$ (I-D requires that implementations do this).

Assumptions

No problems in the random oracle model for KDF. But we can do better.

. . . with several additional assumptions:

- Extract(x, y) is indistinguishable from random function for known x and unknown y, and vice versa.
  - There is no proof of such a property for HKDF-extract(salt, IKM), though it's reasonable to expect such a property.
  - Will try to prove it or make minor changes to the construction.
- Intermediate values of Sig(sk, tag1) are kept secret during the computations (I-D requires that implementations do this).
- sk is never used to sign values from $\mathcal{T}_1$ outside of the construction, limits on $\mathcal{T}_1$ (I-D requires that implementations do this).

Objective: fully specified secure end construction

- Complete security proofs in a very strong adversary model.
- No unreasonable assumptions.

The construction is very specific in the -03 version of the I-D.

The question is: are there additional improvements needed?

- Complete security proof for a KDF with certain security properties, for HKDF we have one additional property to prove.
- Performance issues with HKDF (it is unlikely that it can become a bottleneck, but it must be taken into account).

Obtained results for HKDF

- 16 bytes: $\approx$ 5.3x reduction in performance.
- 256 bytes: $\approx$ 9.9x reduction in performance.
- 8192 bytes: $\approx$ 8.1x reduction in performance.

Objective: fully specified secure end construction

- Complete security proofs in a very strong adversary model.
- No unreasonable assumptions.

The construction is very specific in the -03 version of the I-D.

The question is: are there additional improvements needed?

- Complete security proof for a KDF with certain security properties, for HKDF we have one additional property to prove.
- Performance issues with HKDF (it is unlikely that it can become a bottleneck, but it must be taken into account).

Obtained results for HKDF

- 16 bytes: $\approx$ 5.3x reduction in performance.
- 256 bytes: $\approx$ 9.9x reduction in performance.
- 8192 bytes: $\approx$ 8.1x reduction in performance.

Current state and plans

draft-irtf-cfrg-randomness-improvements
"Randomness Improvements for Security Protocols"

The structure, principles and major recommendations seem to be negotiated and do not tend to be changed.

- New experiments.
- Recommendations for specific protocols.
- Improving the proof to weaken limitations on $\mathcal{T}_1$.
- Refining security proof for using HKDF (or making changes to KDF in the end construction).

Plan: to get a version addressing these issues until IETF 104.

Thank you for your attention!

Questions?

- Materials, questions, comments:
  - cremers@cispa.saarland
  - lgarratt@cisco.com
  - svs@cryptopro.ru
  - nick@cloudflare.com
  - cawood@apple.com

Wrapper generalization

Let G(·) — the output of some CSPRNG. When randomness is needed, instead of x = G(n) use

$$x = \text{Expand}(\text{Extract}(G(L), H(\text{Sig}(sk, tag1))), tag2, n),$$

Moving in -01 from KDF-PRF (-00) to Extract-Expand (e.g., HKDF) to deal with the limit on extracted randomness per invocation.

Tags prevent collisions across private key operations:

- tag1: Constant string bound to a specific device and protocol.
  - Ties the outputs to a particular environment.
  - Sig(sk, tag1) can be cached (but must never be exposed) — for performance reasons, for eliminating additional operations with sk.
- tag2: Dynamic string — timestamp, counter, etc.
  - Ensures that outputs are unique even if the input randomness source degenerates to constant.

Relaxed requirements for a signature scheme

There was a strict requirement in -00 to the signature scheme: „Moreover, Sig MUST be a deterministic signature function, e.g., deterministic ECDSA“.

It has been relaxed, since the digital signature procedure can use its own entropy source: „or use an independent (and completely reliable) entropy source, e.g., if Sig is implemented in an HSM with its own internal trusted entropy source for signature generation.“