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)
Liliya Akhmetzyanova (lah@cryptopro.ru)

CFRG
IETF 104, March 2019, Prague

Brief overview

Motivation

- Most security mechanisms completely depend on randomness.
- But PRNGs can break or contain design flaws/bugs/backdoors.
- E.g. Debian bug, Android's JCA PRNG flaw, Dual_EC_DBRG.
- Difficult to detect; need a safety net to avoid system compromise.
- $\Rightarrow$ provide a solution to improve any call for entropy.

The construction

When randomness is needed, instead of CSPRNG output G(n) use

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

Changes in -04 version: minor changes

Security considerations

"Hence, it is recommended to generate a key specifically for the purposes of the defined construction and not to use it another way."

Terms

For the Expand function a more appropriate term "variable-length output PRF" is now used.

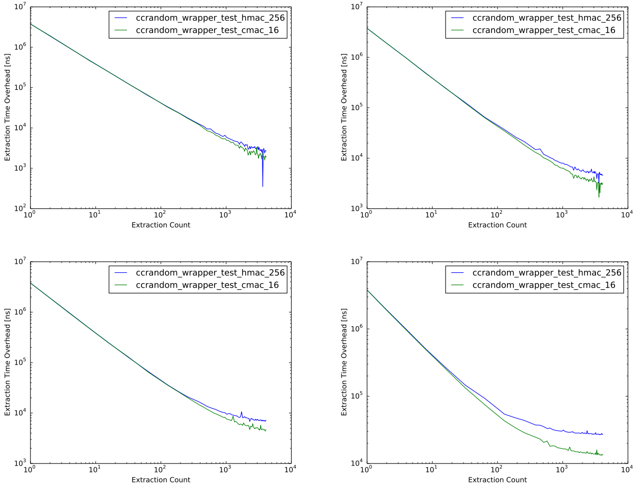
Experimental results

Conditions of the experiment

- Extract and Expand on HKDF (with SHA256) and on CMAC.
- Macbook Pro, a 3.3 GHz Intel Core i7 CPU and 16GB of RAM.
- Compute the signature once, reuse it for all subsequent extractions.
- By varying the number of loops, we can measure the rate at which continued use amortizes the signature computation cost.
- B $= 32, 128, 256, 1024$ output random bytes (common extraction lengths for protocols such as TLS).

$s \leftarrow \mathsf{Hash}(\mathsf{Sign}(\mathsf{sk}, \mathsf{tag}_1))$
for n loops **do**
  $y \leftarrow \mathsf{G}(L)$
  $k \leftarrow \mathsf{Extract}(k, s)$
  $r \leftarrow \mathsf{Expand}(k, \mathsf{tag}_2, B)$
end for

- Collect values of overhead (in comparison with calling only G(L) for the same number of loops).



Observations

- The cost levels out towards an asymptote at approximately 1000 loops.
- At this point, both KDF variants perform nearly equally well.
- Cost: from nanoseconds to microseconds compared to baseline algorithm G.
- Relative cost is minor with respect to cryptographic operations in protocols such as TLS.

Added to the updated eprint.iacr.org/2018/1057 as Appendix.

Summary

- Caching the signature output is critical for performance.
- There's no point for cardinal changes of the construction.
- The construction looks quite well from the performance point of view — keep it as it is.

Security assessment

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 for the initial version (November 2018)

- Complete proofs for all three security properties declared in the draft.
- But, as said at IETF 103, something with the assumptions could still be improved.

Changes in the updated version of the paper

Cryptology ePrint Archive: Report 2018/1057,
https://eprint.iacr.org/2018/1057 — updated in March 2019.

Summary of changes

- Further relaxation of assumptions on the primitives.
- The model reflects real-world conditions and the desired security properties declared in the draft.
- If the adversary can learn outputs of G(L) but cannot control them — a completely finished proof without any questionable assumptions has been obtained.
- New: to address an adversary capable of controlling G(L) we added the analysis for the modified version of the construction (see next two slides).

Strengthening the proof

During the further analysis it was found that one modification of the scheme can provide security in a stricter model, assuming that adversaries can control PRNG outputs.

- In the current version (hashed signature as a "message" input to HMAC) we should expect more from the HMAC.
- If we model HMAC as a random oracle, it won't be a problem.
- But we can do even better.
- It is possible to obtain security proofs even in case of that stronger adversary in a completely honest model.

Proposed change: swapping two inputs of "Extract"

Change

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

to

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

- The modified proof for this has been prepared, addressing also the case of an adversary capable of controlling G(L).
- All other security properties of the construction remain proven.
- Performance increases in default cases: if Extract=HKDF-Extract, we'll be able to cache not only H(Sig(sk, tag1)), but even an intermediate value of HMAC calculation (after processing K $\oplus$ ipad and K $\oplus$ opad during the HMAC calculation).

Current state and plans

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

Only one question left.

- Additional analysis of the proposed change (swapping the "Extract" inputs) will be made.

Plan: to get a version that is ready for a RG Last Call before May.

Backup slides

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 $G(\cdot)$ — the output of some CSPRNG. When randomness is needed, instead of $G(n)$ use

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

Intermediate values (including $G(L)$ and $\mathrm{Sig}(sk, tag1)$) 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.

Experimental results

Comments about signature caching in the I-D

"Sig(sk, tag1) may be cached. In that case the relative cost of using G'(n) instead of G(n) tends to be negligible with respect to cryptographic operations in protocols such as TLS. A description of the performance experiments and their results can be found in the appendix of [SecAnalysis]."

Security assessment

"A security analysis was performed in [SecAnalysis]. Generally speaking, the following security theorem has been proven: 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."

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.

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.“

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).