Randomness Improvements for Security Protocols

draft-irtf-cfrg-randomness-improvements

Cas Cremers (cas.cremers@cs.ox.ac.uk)
Luke Garratt (luke.garratt@cs.ox.ac.uk)
Stanislav Smyshlyaev (svs@cryptopro.ru)
Nick Sullivan (nick@cloudflare.com)
Christopher A. Wood (cawood@apple.com)

CFRG
IETF 102, July 2018, Montreal

Brief overview

Motivation

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

- Debian bug: PRNG seeding process broken by removing crucial mixing step.
- Backdoored DRBGs: Dual_EC_DRBG as an obvious example.
- Any hardware RNG can break someday.
- A joint system entropy pool can be impacted by an attacker.

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

Brief overview

Rationale

- Build on “NAXOS trick” (LaMacchia, Brian et al., “Stronger Security of Authenticated Key Exchange”).
- Mix-in private key in a secure way. But take into account that direct access to private keys is not always possible – and that’s good from the security point of view.
- Consider associated issues (both technical and security-related) and conduct security analysis.
- 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 $x = G(n)$ use

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

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.

Wrapper generalization

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

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

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.
  - $\mathrm{Sig}(sk, tag1)$ 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.

Wrapper generalization

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

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

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.
  - $\mathrm{Sig}(sk, tag1)$ 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.

Defining obtained security properties

The security properties provided by the construction are now (starting from -02) outlined in the draft.

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.

Defining obtained security properties

The security properties provided by the construction are now (starting from -02) outlined in the draft.

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.

Defining obtained security properties

The security properties provided by the construction are now (starting from -02) outlined in the draft.

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.

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.“

Minor issues

Comparison to RFC 6979

Basing on the e-mail thread, paragraphs about similarities/differences with the construction from RFC 6979 have been added (-01, -02): similar constructions, but completely different semantics and obtained security properties.

Example

For example, if in a certain system all private key operations are performed within an HSM, then the differences will manifest as follows: the HMAC DRBG construction of RFC 6979 may be implemented inside the HSM for the sake of signature generation, while the proposed construction would assume calling the signature implemented in the HSM.

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.

- Experiments with existing implementations.
- Clarifying security proofs regarding the three stated properties.
- Further thinking about recommendations for specific protocols to be added.
- Refining security considerations to be stated explicitly, e.g., possibly, „intermediate values (including G(L) and Sig(sk, tag1)) must be kept secret."

Plan: to get a version addressing these issues until IETF 103.

Thank you for your attention!

Questions?

- Materials, questions, comments:
    - cas.cremers@cs.ox.ac.uk
    - luke.garratt@cs.ox.ac.uk
    - svs@cryptopro.ru
    - nick@cloudflare.com
    - cawood@apple.com