Network Working Group   C. Cremers
Internet-Draft   L. Garratt
Intended status: Informational   University of Oxford
Expires: September 12, 2019   S. Smyshlyaev
                                                          CryptoPro
                                                        N. Sullivan
                                                         Cloudflare
                                                           C. Wood
                                                         Apple Inc.
                                                      March 11, 2019

Randomness Improvements for Security Protocols
draft-irtf-cfrg-randomness-improvements-04

Abstract

   Randomness is a crucial ingredient for TLS and related security
   protocols. Weak or predictable "cryptographically-strong"
   pseudorandom number generators (CSPRNGs) can be abused or exploited
   for malicious purposes. The Dual EC random number backdoor and
   Debian bugs are relevant examples of this problem. An initial
   entropy source that seeds a CSPRNG might be weak or broken as well,
   which can also lead to critical and systemic security problems. This
   document describes a way for security protocol participants to
   augment their CSPRNGs using long-term private keys. This improves
   randomness from broken or otherwise subverted CSPRNGs.

Status of This Memo

   This Internet-Draft is submitted in full conformance with the
   provisions of BCP 78 and BCP 79.

   Internet-Drafts are working documents of the Internet Engineering
   Task Force (IETF). Note that other groups may also distribute
   working documents as Internet-Drafts. The list of current Internet-
   Drafts is at https://datatracker.ietf.org/drafts/current/.

   Internet-Drafts are draft documents valid for a maximum of six months
   and may be updated, replaced, or obsoleted by other documents at any
   time. It is inappropriate to use Internet-Drafts as reference
   material or to cite them other than as "work in progress."

   This Internet-Draft will expire on September 12, 2019.

Copyright Notice

Table of Contents

1. Introduction

   Randomness is a crucial ingredient for TLS and related transport
   security protocols. TLS in particular uses random number generators
   (generally speaking, CSPRNGs) to generate several values: session
   IDs, ephemeral key shares, and ClientHello and ServerHello random
   values. CSPRNG failures such as the Debian bug described in
   [DebianBug] can lead to insecure TLS connections. CSPRNGs may also
   be intentionally weakened to cause harm [DualEC]. Initial entropy
   sources can also be weak or broken, and that would lead to insecurity
   of all CSPRNG instances seeded with them. In such cases where
   CSPRNGs are poorly implemented or insecure, an adversary may be able
   to predict its output and recover secret Diffie-Hellman key shares
   that protect the connection.

   This document proposes an improvement to randomness generation in
   security protocols inspired by the "NAXOS trick" [NAXOS].
   Specifically, instead of using raw randomness where needed, e.g., in
   generating ephemeral key shares, a party's long-term private key is
   mixed into the entropy pool. In the NAXOS key exchange protocol, raw

random value x is replaced by H(x, sk), where sk is the sender's
private key. Unfortunately, as private keys are often isolated in
HSMs, direct access to compute H(x, sk) is impossible. An alternate
yet functionally equivalent construction is needed.

The approach described herein replaces the NAXOS hash with a keyed
hash, or pseudorandom function (PRF), where the key is derived from a
raw random value and a private key signature. Implementations SHOULD
apply this technique when indirect access to a private key is
available and CSPRNG randomness guarantees are dubious, or to provide
stronger guarantees about possible future issues with the randomness.
Roughly, the security properties provided by the proposed
construction are as follows:

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.

2. Randomness Wrapper

Let x be the output of a CSPRNG. When properly instantiated, x
should be indistinguishable from a random string of x bytes.
However, as previously discussed, this is not always true. To
mitigate this problem, we propose an approach for wrapping the CSPRNG
output with a construction that mixes secret data into a value that
may be lacking randomness.

Let G(n) be an algorithm that generates n random bytes, i.e., the
output of a CSPRNG. Define an augmented CSPRNG G' as follows. Let
Sig(sk, m) be a function that computes a signature of message m given
private key sk. Let H be a cryptographic hash function that produces
output of length M. Let Extract(salt, IKM) be a randomness
extraction function, e.g., HKDF-Extract [RFC5869], which accepts a
salt and input keying material (IKM) parameter and produces a
pseudorandom key of length L suitable for cryptographic use. Let
Expand(k, info, n) be a variable-length output PRF, e.g., HKDF-Expand
[RFC5869], that takes as input a pseudorandom key k of length L, info

string, and output length n, and produces output of n bytes.
Finally, let tag1 be a fixed, context-dependent string, and let tag2
be a dynamically changing string.

The construction works as follows. Instead of using G(n) when
randomness is needed, use G'(n), where

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

Functionally, this expands n random bytes from a key derived from the
CSPRNG output and signature over a fixed string (tag1). See
Section 3 for details about how "tag1" and "tag2" should be generated
and used per invocation of the randomness wrapper. Expand()
generates a string that is computationally indistinguishable from a
truly random string of n bytes. Thus, the security of this
construction depends upon the secrecy of H(Sig(sk, tag1)) and G(n).
If the signature is leaked, then security of G'(n) reduces to the
scenario wherein randomness is expanded directly from G(n).

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.

Sig(sk, tag1) should only be computed once for the lifetime of the
randomness wrapper, and MUST NOT be used or exposed beyond its role
in this computation. To achieve this, tag1 may have the format that
is not supported (or explicitly forbidden) by other applications
using sk.

Sig MUST be a deterministic signature function, e.g., deterministic
ECDSA [RFC6979], 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.

In systems where signature computations are expensive, 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].

Moreover, the values of G'(n) may be precomputed and pooled. This is
possible since the construction depends solely upon the CSPRNG output
and private key.

3. Tag Generation

   Both tags SHOULD be generated such that they never collide with
   another contender or owner of the private key. This can happen if,
   for example, one HSM with a private key is used from several servers,
   or if virtual machines are cloned.

   To mitigate collisions, tag strings SHOULD be constructed as follows:

   o tag1: Constant string bound to a specific device and protocol in
      use. This allows caching of Sig(sk, tag1). Device specific
      information may include, for example, a MAC address. 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 virtual machines
      (including ones that were cloned or recovered from snapshots). It
      is needed to address the problem of CSPRNG state cloning (see
      [RY2010]). See Section 4 for example protocol information that
      can be used in the context of TLS 1.3.

   o tag2: Non-constant string that includes a timestamp or counter.
      This ensures change over time even if outputs of G(L) were to
      repeat. It 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).

4. Application to TLS

   The PRF randomness wrapper can be applied to any protocol wherein a
   party has a long-term private key and also generates randomness.
   This is true of most TLS servers. Thus, to apply this construction
   to TLS, one simply replaces the "private" CSPRNG G(n), i.e., the
   CSPRNG that generates private values, such as key shares, with:

   G'(n) = HKDF-Expand(HKDF-Extract(G(L), H(Sig(sk, tag1))), tag2, n)

   Moreover, we fix tag1 to protocol-specific information such as "TLS
   1.3 Additional Entropy" for TLS 1.3. Older variants use similarly
   constructed strings.

5. Acknowledgements

   We thank Liliya Akhmetzyanova for her deep involvement in the
   security assessment in [SecAnalysis].

6. IANA Considerations

   This document makes no request to IANA.

7. Security Considerations

   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.

   The main reason one might expect the signature to be exposed is via a
   side-channel attack. It is therefore prudent when implementing this
   construction to take into consideration the extra long-term key
   operation if equipment is used in a hostile environment when such
   considerations are necessary. Hence, it is recommended to generate a
   key specifically for the purposes of the defined construction and not
   to use it another way.

   The signature in the construction as well as in the protocol itself
   MUST NOT use randomness from entropy sources with dubious security
   guarantees. Thus, the signature scheme MUST either use a reliable
   entropy source (independent from the CSPRNG that is being improved
   with the proposed construction) or be deterministic: if the
   signatures are probabilistic and use weak entropy, our construction
   does not help and the signatures are still vulnerable due to repeat
   randomness attacks. In such an attack, the adversary might be able
   to recover the long-term key used in the signature.

   Under these conditions, applying this construction should never yield
   worse security guarantees than not applying it assuming that applying
   the PRF does not reduce entropy. We believe there is always merit in
   analyzing protocols specifically. However, this construction is
   generic so the analyses of many protocols will still hold even if
   this proposed construction is incorporated.

   The proposed construction cannot provide any guarantees of security
   if the CSPRNG state is cloned due to the virtual machine snapshots or
   process forking (see [MAFS2017]). Thus tag1 SHOULD incorporate all
   available information about the environment, such as process
   attributes, virtual machine user information, etc.

8. Comparison to RFC 6979

   The construction proposed herein has similarities with that of RFC
   6979 [RFC6979]: both of them use private keys to seed a DRBG.
   Section 3.3 of RFC 6979 recommends deterministically instantiating an
   instance of the HMAC DRBG pseudorandom number generator, described in
   [SP80090A] and Annex D of [X962], using the private key sk as the
   entropy_input parameter and H(m) as the nonce. The construction
   G'(n) provided herein is similar, with such difference that a key
   derived from G(n) and H(Sig(sk, tag1)) is used as the entropy input
   and tag2 is the nonce.

   However, the semantics and the security properties obtained by using
   these two constructions are different. The proposed construction
   aims to improve CSPRNG usage such that certain trusted randomness
   would remain even if the CSPRNG is completely broken. Using a
   signature scheme which requires entropy sources according to RFC 6979
   is intended for different purposes and does not assume possession of
   any entropy source - even an unstable one. 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.

9. Normative References

   [DebianBug]
             Yilek, Scott, et al, ., "When private keys are public -
             Results from the 2008 Debian OpenSSL vulnerability", n.d.,
             <https://pdfs.semanticscholar.org/fcf9/
             fe0946c20e936b507c023bbf89160cc995b9.pdf>.

   [DualEC]   Bernstein, Daniel et al, ., "Dual EC - A standardized back
             door", n.d., <https://projectbullrun.org/dual-
             ec/documents/dual-ec-20150731.pdf>.

   [MAFS2017]
             McGrew, Anderson, Fluhrer, Shenefeil, ., "PRNG Failures
             and TLS Vulnerabilities in the Wild", n.d.,
             <https://rwc.iacr.org/2017/Slides/david.mcgrew.pptx>.

   [NAXOS]   LaMacchia, Brian et al, ., "Stronger Security of
             Authenticated Key Exchange", n.d.,
             <https://www.microsoft.com/en-us/research/wp-
             content/uploads/2016/02/strongake-submitted.pdf>.

   [RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-
              Hashing for Message Authentication", RFC 2104,
              DOI 10.17487/RFC2104, February 1997,
              <https://www.rfc-editor.org/info/rfc2104>.

   [RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand
              Key Derivation Function (HKDF)", RFC 5869,
              DOI 10.17487/RFC5869, May 2010,
              <https://www.rfc-editor.org/info/rfc5869>.

   [RFC6979] Pornin, T., "Deterministic Usage of the Digital Signature
              Algorithm (DSA) and Elliptic Curve Digital Signature
              Algorithm (ECDSA)", RFC 6979, DOI 10.17487/RFC6979, August
              2013, <https://www.rfc-editor.org/info/rfc6979>.

   [RY2010]   Ristenpart, Yilek, ., "When Good Randomness Goes Bad|:|
              Virtual Machine Reset Vulnerabilities and Hedging Deployed
              Cryptography", n.d.,
              <https://rist.tech.cornell.edu/papers/sslhedge.pdf>.

   [SecAnalysis]
              Akhmetzyanova, Cremers, Garratt, Smyshlyaev, ., "Security
              Analysis for Randomness Improvements for Security
              Protocols", n.d., <https://eprint.iacr.org/2018/1057>.

   [SP80090A]
              "Recommendation for Random Number Generation Using
              Deterministic Random Bit Generators (Revised), NIST
              Special Publication 800-90A, January 2012.", n.d.,
              <National Institute of Standards and Technology>.

   [X9.62]   American National Standards Institute, ., "Public Key
              Cryptography for the Financial Services Industry -- The
              Elliptic Curve Digital Signature Algorithm (ECDSA). ANSI
              X9.62-2005, November 2005.", n.d..

   [X962]   "Public Key Cryptography for the Financial Services
              Industry -- The Elliptic Curve Digital Signature Algorithm
              (ECDSA), ANSI X9.62-2005, November 2005.", n.d., <American
              National Standards Institute>.

Authors' Addresses

Cas Cremers
University of Oxford
Wolfson Building, Parks Road
Oxford
England

Email: cas.cremers@cs.ox.ac.uk

Luke Garratt
University of Oxford
Wolfson Building, Parks Road
Oxford
England

Email: luke.garratt@cs.ox.ac.uk

Stanislav Smyshlyaev
CryptoPro
18, Suschevsky val
Moscow
Russian Federation

Email: svs@cryptopro.ru

Nick Sullivan
Cloudflare
101 Townsend St
San Francisco
United States of America

Email: nick@cloudflare.com

Christopher A. Wood
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America

Email: cawood@apple.com