ï»¿

Internet Research Task Force (IRTF)
Request for Comments: 8937
Category: Informational
ISSN: 2070-1721

C. Cremers
CISPA
L. Garratt
Cisco Meraki
S. Smyshlyaev
CryptoPro
N. Sullivan
C. Wood
Cloudflare

October 2020

Randomness Improvements for Security Protocols

Abstract

Randomness is a crucial ingredient for Transport Layer Security (TLS) and related security protocols. Weak or predictable "cryptographically secure" pseudorandom number generators (CSPRNGs) can be abused or exploited for malicious purposes. 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 implementations to augment their CSPRNGs using long-term private keys. This improves randomness from broken or otherwise subverted CSPRNGs.

This document is a product of the Crypto Forum Research Group (CFRG) in the IRTF.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Research Task Force (IRTF). The IRTF publishes the results of Internet-related research and development activities. These results might not be suitable for deployment. This RFC represents the consensus of the Crypto Forum Research Group of the Internet Research Task Force (IRTF). Documents approved for publication by the IRSG are not a candidate for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8937.

Table of Contents

1. Introduction

Secure and properly implemented random number generators, or "cryptographically secure" pseudorandom number generators (CSPRNGs), should produce output that is indistinguishable from a random string of the same length. CSPRNGs are critical building blocks for TLS and related transport security protocols. TLS in particular uses CSPRNGs to generate several values, such as 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, Adv, may be able to distinguish its output from a random string or predict its output and recover secret key material used to 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 function of 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 Hardware Security Modules (HSMs), direct access to compute H(x, sk) is impossible. Moreover, some HSM APIs may only offer the option to sign messages using a private key, yet offer no other operations involving that key. 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 a) when indirect access to a private key is available and CSPRNG randomness guarantees are dubious or b) 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. Adv with full control of a (potentially broken) CSPRNG and ability to observe all outputs of the proposed construction does not obtain any non-negligible advantage in leaking the private key (in the absence of side channel attacks).

3. If the CSPRNG is broken or controlled by Adv, the output of the proposed construction remains indistinguishable from random, provided that the private key remains unknown to Adv.

This document represents the consensus of the Crypto Forum Research Group (CFRG).

2. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in

BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Randomness Wrapper

The output of a properly instantiated CSPRNG should be indistinguishable from a random string of the same length. 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 L bytes suitable for cryptographic use. It must be a secure PRF (for salt as a key of length M) and preserve uniformness of IKM (for details, see [SecAnalysis]). L SHOULD be a fixed length. 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 L bytes, 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 (e.g., a counter) of L' bytes. We require that L >= n - L' for each value of tag2.

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

```
G'(n) = Expand(Extract(H(Sig(sk, tag1)), G(L)), 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 4 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(L). If the signature is leaked, then security of G'(n) reduces to the scenario wherein randomness is expanded directly from G(L).

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 function, and Expand function) can be implemented either inside or outside the HSM.

Sig(sk, tag1) need only be computed once for the lifetime of the randomness wrapper and MUST NOT be used or exposed beyond its role in this computation. Additional recommendations for tag1 are given in the following section.

Sig MUST be a deterministic signature function, e.g., deterministic Elliptic Curve Digital Signature Algorithm (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.

Because Sig(sk, tag1) can be cached, 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 (the relatively inexpensive computational cost of HKDF-Extract and HKDF-Expand dominates when comparing G' to G). A description of the performance experiments and their results can be found in [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.

4. Tag Generation

Both tags MUST 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.

The RECOMMENDED tag construction procedure is as follows:

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 Media Access Control (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). This is needed to address the problem of CSPRNG state cloning (see [RY2010]). See Section 5 for example protocol information that can be used in the context of TLS 1.3. If sk could be used for other purposes, then selecting a value for tag1 that is different than the form allowed by those other uses ensures that the signature is not exposed.

tag2: A nonce, that is, a value that is unique for each use of the same combination of G(L), tag1, and sk values. The tag2 value can be implemented using a counter or a timer, provided that the timer is guaranteed to be different for each invocation of G'(n).

5. 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(H(Sig(sk, tag1)), G(L)), tag2, n)

6. Implementation Guidance

Recall that the wrapper defined in Section 3 requires L >= n - L', where L is the Extract output length and n is the desired amount of randomness. Some applications may require n to exceed this bound. Wrapper implementations can support this use case by invoking G' multiple times and concatenating the results.

7. IANA Considerations

This document has no IANA actions.

8. Security Considerations

A security analysis was performed in [SecAnalysis]. Generally speaking, the following security theorem has been proven: if Adv 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, Adv 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]). It is RECOMMENDED that tag1 incorporate all available information about the environment, such as process attributes, virtual machine user information, etc.

9. Comparison to RFC 6979

The construction proposed herein has similarities with that of [RFC6979]; both of them use private keys to seed a deterministic random number generator. Section 3.3 of [RFC6979] 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 that requires entropy sources according to [RFC6979] 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 described in [RFC6979] 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.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[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>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

10.2. Informative References

[DebianBug]
Yilek, S., Rescorla, E., Shacham, H., Enright, B., and S. Savage, "When private keys are public: results from the 2008 Debian OpenSSL vulnerability", ICM '09, DOI 10.1145/1644893.1644896, November 2009, <https://pdfs.semanticscholar.org/fcf9/fe0946c20e936b507c023bbf89160cc995b9.pdf>.

[DualEC] Bernstein, D., Lange, T., and R. Niederhagen, "Dual EC: A Standardized Back Door", DOI 10.1007/978-3-662-49301-4_17, March 2016, <https://projectbullrun.org/dual-ec/documents/dual-ec-20150731.pdf>.

[MAFS2017] McGrew, D., Anderson, B., Fluhrer, S., and C. Schenefiel, "PRNG Failures and TLS Vulnerabilities in the Wild", January 2017, <https://rwc.iacr.org/2017/Slides/david.mcgrew.pptx>.

[NAXOS] LaMacchia, B., Lauter, K., and A. Mityagin, "Stronger Security of Authenticated Key Exchange", DOI 10.1007/978-3-540-75670-5_1, November 2007, <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/strongake-submitted.pdf>.

[RY2010] Ristenpart, T. and S. Yilek, "When Good Randomness Goes Bad: Virtual Machine Reset Vulnerabilities and Hedging Deployed Cryptography", January 2010, <https://rist.tech.cornell.edu/papers/sslhedge.pdf>.

[SecAnalysis]
Akhmetzyanova, L., Cremers, C., Garratt, L., Smyshlyaev, S., and N. Sullivan, "Limiting the impact of unreliable randomness in deployed security protocols", DOI 10.1109/CSF49147.2020.00027, IEEE 33rd Computer Security Foundations Symposium (CSF), Boston, MA, USA, pp. 385-393, 2020, <https://doi.org/10.1109/CSF49147.2020.00027>.

[SP80090A] National Institute of Standards and Technology, "Recommendation for Random Number Generation Using Deterministic Random Bit Generators, Special Publication 800-90A Revision 1", DOI 10.6028/NIST.SP.800-90Ar1, June 2015, <https://doi.org/10.6028/NIST.SP.800-90Ar1>.

[X962] American National Standard for Financial Services (ANSI), "Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)", ANSI X9.62, November 2005, <https://www.techstreet.com/standards/x9-x9-62-2005?product_id=1327225>.

Acknowledgements

We thank Liliya Akhmetzyanova for her deep involvement in the security assessment in [SecAnalysis]. We thank John Mattsson, Martin Thomson, and Rich Salz for their careful readings and useful comments.

Authors' Addresses

Cas Cremers
CISPA
Saarland Informatics Campus
Saarbruecken
Germany

Email: cremers@cispa.saarland

Luke Garratt
Cisco Meraki
500 Terry A Francois Blvd
San Francisco,
United States of America

Email: lgarratt@cisco.com

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
Cloudflare
101 Townsend St
San Francisco,
United States of America

Email: caw@heapingbits.net