

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 6 August 2022

F. Denis
Fastly Inc.
F. Jacobs
Apple Inc.
C.A. Wood
Cloudflare
2 February 2022

RSA Blind Signatures
draft-irtf-cfrg-rsa-blind-signatures-03

Abstract

This document specifies the RSA-based blind signature protocol with appendix (RSA-BSSA). RSA blind signatures were first introduced by Chaum for untraceable payments [[Chaum83](#)]. It extends RSA-PSS encoding specified in [[RFC8017](#)] to enable blind signature support.

Discussion Venues

This note is to be removed before publishing as an RFC.

Source for this draft and an issue tracker can be found at <https://github.com/chris-wood/draft-wood-cfrg-blind-signatures>.

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 6 August 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the [Trust Legal Provisions](#) and are provided without warranty as described in the Revised BSD License.

Table of Contents

1.	Introduction	2
2.	Requirements Notation	3
3.	Notation	3
4.	Blind Signature Protocol Overview	4
5.	RSABSSA Signature Instantiation	4
5.1.	Signature Generation	4
5.1.1.	Blind	5
5.1.2.	BlindSign	6
5.1.3.	Finalize	6
5.2.	Encoding Options	7
6.	Public Key Certification	8
7.	Implementation Considerations	8
7.1.	Errors	8
7.2.	API Considerations	8
8.	Security Considerations	9
8.1.	Timing Side Channels	9
8.2.	Message Robustness	9
8.3.	Randomized and Deterministic Signatures	10
8.4.	Key Substitution Attacks	10
8.5.	Alternative RSA Encoding Functions	10
8.6.	Alternative Blind Signature Protocols	11
8.7.	Post-Quantum Readiness	12
9.	IANA Considerations	12
10.	References	12
10.1.	Normative References	12
10.2.	Informative References	13
Appendix A.	Test Vectors	16
	Authors' Addresses	21

[1.](#) Introduction

Originally introduced in the context of digital cash systems by Chaum

for untraceable payments [[Chaum83](#)], RSA blind signatures turned out to have a wide range of applications ranging from electric voting schemes to authentication mechanisms.

Recently, interest in blind signatures has grown to address operational shortcomings from VOPRFs such as [[I-D.irtf-cfrg-voprf](#)]. Specifically, VOPRF evaluation requires access to the private key, and is therefore required for both issuance and redemption of tokens in anonymous authentication protocols such as Privacy Pass [[I-D.davidson-pp-protocol](#)]. This limitation complicates deployments where it is not desirable to distribute secret keys to entities performing token verification. Additionally, if the private key is kept in a Hardware Security Module, the number of operations on the key is doubled compared to a scheme where the private key is only required for issuance of the tokens.

In contrast, cryptographic signatures provide a primitive that is publicly verifiable and does not require access to the private key for verification. Moreover, [[JKK14](#)] shows that one can realize a VOPRF in the Random Oracle Model by hashing a (deterministic) blind signature-message pair.

This document specifies a protocol for the RSA Blind Signature Scheme with Appendix (RSABSSA). In order to facilitate deployment, we define it in such a way that the resulting (unblinded) signature can be verified with a standard RSA-PSS library.

[2.](#) Requirements Notation

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.](#) Notation

The following terms are used throughout this document to describe the protocol operations in this document:

- * `bytes_to_int` and `int_to_bytes`: Convert a byte string to and from a non-negative integer. `bytes_to_int` and `int_to_bytes` are implemented as OS2IP and I2OSP as described in [[RFC8017](#)], respectively. Note that these functions operate on byte strings in big-endian byte order.
- * `random_integer_uniform(M, N)`: Generate a random, uniformly distributed integer R such that $M \leq R < N$.
- * `inverse_mod(x, n)`: Compute the multiplicative inverse of $x \bmod n$. This function fails if x and n are not co-prime.

- * `len(s)`: The length of a byte string, in octets.

[4.](#) Blind Signature Protocol Overview

In this section, we sketch the blind signature protocol wherein a client and server interact to compute $\text{sig} = \text{Sign}(\text{skS}, \text{msg})$, where `msg` is the private message to be signed, and `skS` is the server's private key. In this protocol, the server learns nothing of `msg`, whereas the client learns `sig` and nothing of `skS`.

The core issuance protocol runs as follows:

```

Client(pkS, msg)                                Server(skS, pkS)
-----
blinded_msg, inv = Blind(pkS, msg)

                                blinded_msg
                                ----->

                                blind_sig = BlindSign(skS, blinded_msg)

                                blind_sig
                                <-----

sig = Finalize(pkS, msg, blind_sig, inv)
```

Upon completion, correctness requires that clients can verify signature `sig` over private input message `msg` using the server public key `pkS` by invoking the RSASSA-PSS-VERIFY routine defined in

[RFC3447]. The finalization function performs that check before returning the signature.

5. RSABSSA Signature Instantiation

[Section 8.1 of \[RFC8017\]](#) defines RSASSA-PSS RSAE, which is a signature algorithm using RSASSA-PSS [\[RFC8017\]](#) with mask generation function 1. In this section, we define RSABSSA, a blinded variant of this algorithm.

5.1. Signature Generation

As outlined in [Section 4](#), signature generation involves three subroutines: Blind, BlindSign, and Finalize. The output from Finalize is a signature over the input to Blind. A specification of these subroutines is below.

5.1.1. Blind

rsabssa_blind encodes an input message and blinds it with the server's public key. It outputs the blinded message to be sent to the server and the corresponding inverse, both encoded as octet strings. RSAVP1 and EMSA-PSS-ENCODE are as defined in [\[RFC3447\]](#).

rsabssa_blind(pkS, msg)

Parameters:

- kLen, the length in octets of the RSA modulus n
- kBits, the length in bits of the RSA modulus n
- HF, the hash function used to hash the message
- MGF, the mask generation function

Inputs:

- pkS, server public key (n , e)
- msg, message to be signed, an octet string

Outputs:

- blinded_msg, an octet string of length kLen
- inv, an octet string of length kLen

Errors:

- "message too long": Raised when the input message is too long.
- "encoding error": Raised when the input message fails encoding.
- "invalid blind": Raised when the inverse of r cannot be found.

Steps:

1. `encoded_msg = EMSA-PSS-ENCODE(msg, kBits - 1)`
with MGF and HF as defined in the parameters
2. If EMSA-PSS-ENCODE raises an error, raise the error and stop
3. `m = bytes_to_int(encoded_msg)`
4. `r = random_integer_uniform(1, n)`
5. `r_inv = inverse_mod(r, n)`
6. If `inverse_mod` fails, raise an "invalid blind" error and stop
7. `x = RSAVP1(pkS, r)`
8. `z = m * x mod n`
9. `blinded_msg = int_to_bytes(z, kLen)`
10. `inv = int_to_bytes(r_inv, kLen)`
11. output `blinded_msg, inv`

The blinding factor r must be randomly chosen from a uniform distribution. This is typically done via rejection sampling.

[5.1.2.](#) BlindSign

`rsabssa_blind_sign` performs the RSA private key operation on the client's blinded message input and returns the output encoded as an octet string. RSASP1 is as defined in [\[RFC3447\]](#).

`rsabssa_blind_sign(skS, blinded_msg)`

Parameters:

- `kLen`, the length in octets of the RSA modulus n

Inputs:

- `skS`, server private key
- `blinded_msg`, encoded and blinded message to be signed, an octet string

Outputs:

- blind_sig, an octet string of length kLen

Errors:

- "unexpected input size": Raised when a byte string input doesn't have the expected length.
- "invalid message length": Raised when the message representative to sign is not an integer between 0 and $n - 1$.

Steps:

1. If `len(blinded_msg) != kLen`, raise "unexpected input size" and stop
2. `m = bytes_to_int(blinded_msg)`
3. If `m >= n`, raise "invalid message length" and stop
4. `s = RSASP1(skS, m)`
5. `blind_sig = int_to_bytes(s, kLen)`
6. output blind_sig

[5.1.3.](#) Finalize

`rsabssa_finalize` validates the server's response, unblinds the message to produce a signature, verifies it for correctness, and outputs the signature upon success. Note that this function will internally hash the input message as is done in `rsabssa_blind`.

`rsabssa_finalize(pkS, msg, blind_sig, inv)`

Parameters:

- kLen, the length in octets of the RSA modulus n

Inputs:

- pkS, server public key (n, e)
- msg, message to be signed, an octet string

- blind_sig, signed and blinded element, an octet string of length kLen
- inv, inverse of the blind, an octet string of length kLen

Outputs:

- sig, an octet string of length kLen

Errors:

- "invalid signature": Raised when the signature is invalid
- "unexpected input size": Raised when a byte string input doesn't have the expected length.

Steps:

1. If $\text{len}(\text{blind_sig}) \neq \text{kLen}$, raise "unexpected input size" and stop
2. If $\text{len}(\text{inv}) \neq \text{kLen}$, raise "unexpected input size" and stop
3. $z = \text{bytes_to_int}(\text{blind_sig})$
4. $r_inv = \text{bytes_to_int}(\text{inv})$
5. $s = z * r_inv \bmod n$
6. $\text{sig} = \text{int_to_bytes}(s, \text{kLen})$
7. $\text{result} = \text{RSASSA-PSS-VERIFY}(\text{pkS}, \text{msg}, \text{sig})$
8. If $\text{result} = \text{"valid signature"}$, output sig, else raise "invalid signature" and stop

5.2. Encoding Options

The RSASSA-PSS parameters, defined as in [\[RFC8017\], Section 9.1.1](#), are as follows:

- * Hash: hash function (hLen denotes the length in octets of the hash function output)
- * MGF: mask generation function
- * sLen: intended length in octets of the salt

It is RECOMMENDED that implementations support the following encoding options:

- * SHA-384 as Hash and MGF functions and sLen = 48, as described in [\[RFC8230\], Section 2](#); and

- * SHA-384 as Hash and MGF functions and sLen = 0.

Note that setting `sLen = 0` has the result of making the signature deterministic.

The blinded functions in [Section 5.1](#) are orthogonal to the choice of these encoding options.

[6.](#) Public Key Certification

If the server public key is carried in an X.509 certificate, it MUST use the RSASSA-PSS OID [[RFC5756](#)]. It MUST NOT use the `rsaEncryption` OID [[RFC5280](#)].

[7.](#) Implementation Considerations

This section documents considerations for interfaces to implementations of the protocol in this document. This includes error handling and API considerations.

[7.1.](#) Errors

The high-level functions specified in [Section 5.1](#) are all fallible. The explicit errors generated throughout this specification, along with the conditions that lead to each error, are listed in the definitions for `rsabssa_blind`, `rsabssa_blind_sign`, and `rsabssa_finalize`. These errors are meant as a guide for implementors. They are not an exhaustive list of all the errors an implementation might emit. For example, implementations might run out of memory.

[7.2.](#) API Considerations

It is NOT RECOMMENDED that APIs allow clients to specify RSA-PSS parameters directly, e.g., to set the PSS salt value or its length. Instead, it is RECOMMENDED that implementations generate the PSS salt using the same source of randomness used to produce the blinding factor.

If implementations need support for randomized and deterministic signatures, they should offer separate abstractions for each. Allowing callers to control the PSS salt value or length may have security consequences. See [Section 8.3](#) for more information about details.

[8.](#) Security Considerations

Bellare et al. [[BNPS03](#)] proved the following properties of Chaum's original blind signature protocol based on RSA-FDH:

- * One-more-forgery polynomial security. This means the adversary, interacting with the server (signer) as a client, cannot output $n+1$ valid message and signature tuples after only interacting with the server n times, for some n which is polynomial in the protocol's security parameter.
- * Concurrent polynomial security. This means that servers can engage in polynomially many invocations of the protocol without compromising security.

Both results rely upon the RSA Known Target Inversion Problem being hard.

The design in this document differs from the analysis in [[BNPS03](#)] only in message encoding, i.e., using PSS instead of FDH. Note, importantly, that an empty salt effectively reduces PSS to FDH, so the same results apply.

[8.1.](#) Timing Side Channels

`rsabssa_blind_sign` is functionally a remote procedure call for applying the RSA private key operation. As such, side channel resistance is paramount to protect the private key from exposure [[RemoteTimingAttacks](#)]. Implementations MUST implement RSA blinding as a side channel attack mitigation. One mechanism is described in Section 10 of [[TimingAttacks](#)]. Failure to do so may lead to side channel attacks that leak the private signing key.

[8.2.](#) Message Robustness

An essential property of blind signature protocols is that the signer learns nothing of the message being signed. In some circumstances, this may raise concerns of arbitrary signing oracles. Applications using blind signature protocols should take precautions to ensure that such oracles do not cause cross-protocol attacks. This can be done, for example, by keeping blind signature keys distinct from signature keys used for other protocols, such as TLS.

An alternative solution to this problem of message blindness is to give signers proof that the message being signed is well-structured. Depending on the application, zero knowledge proofs could be useful

for this purpose. Defining such a proof is out of scope for this document.

Verifiers should check that, in addition to signature validity, the unblinded message is well-structured for the relevant application. For example, if an application of this protocol requires messages to be structures of a particular form, then verifiers should check that unblinded messages adhere to this form.

[8.3.](#) Randomized and Deterministic Signatures

When $sLen > 0$, the PSS salt is a randomly generated string chosen when a message is encoded. This means the resulting signature is non-deterministic. As a result, two signatures over the same message will be different. If the salt is not generated randomly, or is otherwise constructed maliciously, it might be possible for the salt to encode information that is not present in the signed message. For example, the salt might be maliciously constructed to encode the local IP address of the client. As a result, APIs SHOULD NOT allow clients to provide the salt directly; see [Section 7.2](#) for API considerations.

When $sLen = 0$, the PSS salt is empty and the resulting signature is deterministic. Such signatures may be useful for applications wherein the only desired source of entropy is the input message.

Applications that use deterministic signatures SHOULD carefully analyze the security implications. When the required signature protocol is not clear, applications SHOULD default to randomized signatures.

[8.4.](#) Key Substitution Attacks

RSA is well known to permit key substitution attacks, wherein an attacker generates a key pair (skA , pkA) that verify some known (message, signature) pair produced under a different (skS , pkS) key pair [[WM99](#)]. This means it may be possible for an attacker to use a (message, signature) pair from one context in another. Entities that verify signatures must take care to ensure a (message, signature) pair verifies with a valid public key from the expected issuer.

[8.5.](#) Alternative RSA Encoding Functions

This document document uses PSS encoding as specified in [[RFC3447](#)] for a number of reasons. First, it is recommended in recent standards, including TLS 1.3 [[RFC8446](#)], X.509v3 [[RFC4055](#)], and even PKCS#1 itself. According to [[RFC3447](#)], "Although no attacks are known against RSASSA-PKCS#1 v1.5, in the interest of increased robustness, RSA-PSS is recommended for eventual adoption in new applications." While RSA-PSS is more complex than RSASSA-PKCS#1 v1.5 encoding, ubiquity of RSA-PSS support influenced the design decision

in this draft, despite PKCS#1 v1.5 having equivalent security properties for digital signatures [[JKM18](#)]

Full Domain Hash (FDH) [[RSA-FDH](#)] encoding is also possible, and this variant has equivalent security to PSS [[KK18](#)]. However, FDH is less standard and not used widely in related technologies. Moreover, FDH is deterministic, whereas PSS supports deterministic and probabilistic encodings.

[8.6](#). Alternative Blind Signature Protocols

RSA has some advantages as a signature protocol, particularly around verification efficiency. However, the protocol in this document is not without shortcomings, including:

- * RSA key and signature sizes are larger than those of alternative blind signature protocols;
- * No evaluation batching support, which means that the cost of the protocol scales linearly with the number of invocations; and
- * Extensions for features such as threshold signing are more complex to instantiate compared to other protocols based on, for example, Schnorr signatures.

There are a number of blind signature protocols beyond blind RSA. This section summarizes these at a high level, and discusses why an RSA-based variant was chosen for the basis of this specification, despite the shortcomings above.

- * Blind Schnorr [[Sch01](#)]: This is a three-message protocol based on the classical Schnorr signature protocol over elliptic curve

groups. Although simple, the hardness problem upon which this is based -- Random inhomogeneities in a Overdetermined Solvable system of linear equations, or ROS -- can be broken in polynomial time when a small number of concurrent signing sessions are invoked [[PolytimeROS](#)], leading to signature forgeries. Even with small concurrency limits, Wagner's generalized attack [[Wagner02](#)] leads to subexponential forgery speedup. For example, a limit of 15 parallel sessions yields an attack runtime of approximately 2^{55} , which is substantially lower than acceptable security levels. In contrast, the variant in this specification has no such concurrency limit.

- * Clause Blind Schnorr [[FPS20](#)]: This is a three-message protocol based on a variant of the blind Schnorr signature protocol. This variant of the protocol is not known to be vulnerable to the attack in [[PolytimeROS](#)], though the protocol is still new and

under consideration. In the future, this may be a candidate for future blind signatures based on blind signatures. However, the three-message flow necessarily requires two round trips between the client and server, which may be prohibitive for large-scale signature generation. Further analysis and experimentation with this protocol is needed.

- * BSA [[Abe01](#)]: This is a three-message protocol based on elliptic curve groups similar to blind Schnorr. It is also not known to be vulnerable to the ROS attack in [[PolytimeROS](#)]. Kastner et al. [[KLRX20](#)] proved concurrent security with a polynomial number of sessions. For similar reasons to the clause blind Schnorr protocol above, the additional number of round trips requires further analysis and experimentation.
- * Blind BLS [[BLS-Proposal](#)]: The Boneh-Lynn-Shacham [[I-D.irtf-cfrg-bls-signature](#)] protocol can incorporate message blinding when properly instantiated with Type III pairing group. This is a two-message protocol similar to the RSA variant, though it requires pairing support, which is not common in widely deployed cryptographic libraries backing protocols such as TLS. In contrast, the specification in this document relies upon widely deployed cryptographic primitives.

Beyond blind signature protocols, anonymous credential schemes with

public verifiability such as U-Prove [[UProve](#)] may be used instead of blind signature protocols. Anonymous credentials may even be constructed with blind signature protocols. However, anonymous credentials are higher-level constructions that present a richer feature set.

[8.7.](#) Post-Quantum Readiness

The blind signature protocol specified in this document is not post-quantum ready since it is based on RSA. (Shor's polynomial-time factorization algorithm readily applies.)

[9.](#) IANA Considerations

This document makes no IANA requests.

[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://doi.org/10.17487/RFC2119>>.
- [RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", [RFC 3447](#), DOI 10.17487/RFC3447, February 2003, <<https://doi.org/10.17487/RFC3447>>.
- [RFC5756] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Updates for RSAES-OAEP and RSASSA-PSS Algorithm Parameters", [RFC 5756](#), DOI 10.17487/RFC5756, January 2010, <<https://doi.org/10.17487/RFC5756>>.
- [RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", [RFC 8017](#), DOI 10.17487/RFC8017, November 2016,

<<https://doi.org/10.17487/RFC8017>>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in [RFC 2119](#) Key Words", [BCP 14](#), [RFC 8174](#), DOI 10.17487/RFC8174, May 2017, <<https://doi.org/10.17487/RFC8174>>.

[RFC8230] Jones, M., "Using RSA Algorithms with CBOR Object Signing and Encryption (COSE) Messages", [RFC 8230](#), DOI 10.17487/RFC8230, September 2017, <<https://doi.org/10.17487/RFC8230>>.

[10.2](#). Informative References

[Abe01] Abe, M., "A Secure Three-Move Blind Signature Scheme for Polynomially Many Signatures", DOI 10.1007/3-540-44987-6_9, Lecture Notes in Computer Science pp. 136-151, 2001, <https://doi.org/10.1007/3-540-44987-6_9>.

[BLS-Proposal]
"[Privacy-pass] External verifiability: a concrete proposal", July 2020, <<https://mailarchive.ietf.org/arch/msg/privacy-pass/BD00hSLwB3uUJcfBiss6nUF5sUA/>>.

[BNPS03] Bellare, ., Namprempre, ., Pointcheval, ., and . Semanko, "The One-More-RSA-Inversion Problems and the Security of Chaum's Blind Signature Scheme", DOI 10.1007/s00145-002-0120-1, Journal of Cryptology Vol. 16, pp. 185-215, June 2003, <<https://doi.org/10.1007/s00145-002-0120-1>>.

[Chaum83] "Blind Signatures for Untraceable Payments", 1983, <<http://scweb.sce.uhcl.edu/yang/teaching/csci5234WebSecurityFall2011/Chaum-blind-signatures.PDF>>.

- [FPS20] Fuchsbauer, G., Plouviez, A., and Y. Seurin, "Blind Schnorr Signatures and Signed ElGamal Encryption in the Algebraic Group Model", DOI 10.1007/978-3-030-45724-2_3, Advances in Cryptology - EUROCRYPT 2020 pp. 63-95, 2020, <https://doi.org/10.1007/978-3-030-45724-2_3>.
- [I-D.davidson-pp-protocol] Davidson, A., "Privacy Pass: The Protocol", Work in Progress, Internet-Draft, [draft-davidson-pp-protocol-01](https://datatracker.ietf.org/doc/html/draft-davidson-pp-protocol-01), 13 July 2020, <<https://datatracker.ietf.org/doc/html/draft-davidson-pp-protocol-01>>.
- [I-D.irtf-cfrg-bls-signature] Boneh, D., Gorbunov, S., Wahby, R. S., Wee, H., and Z. Zhang, "BLS Signatures", Work in Progress, Internet-Draft, [draft-irtf-cfrg-bls-signature-04](https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-bls-signature-04), 10 September 2020, <<https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-bls-signature-04>>.
- [I-D.irtf-cfrg-voprf] Davidson, A., Faz-Hernandez, A., Sullivan, N., and C. A. Wood, "Oblivious Pseudorandom Functions (OPRFs) using Prime-Order Groups", Work in Progress, Internet-Draft, [draft-irtf-cfrg-voprf-08](https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-voprf-08), 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-voprf-08>>.
- [JKK14] "Round-Optimal Password-Protected Secret Sharing and T-PAKE in the Password-Only model", August 2014, <<https://eprint.iacr.org/2014/650>>.
- [JKM18] Jager, T., Kakvi, S., and A. May, "On the Security of the PKCS#1 v1.5 Signature Scheme", DOI 10.1145/3243734.3243798, Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, January 2018, <<https://doi.org/10.1145/3243734.3243798>>.

- [KK18] Kakvi, S. and E. Kiltz, "Optimal Security Proofs for Full Domain Hash, Revisited", DOI 10.1007/s00145-017-9257-9, Journal of Cryptology Vol. 31, pp. 276-306, April 2017, <<https://doi.org/10.1007/s00145-017-9257-9>>.

- [KLRX20] "On Pairing-Free Blind Signature Schemes in the Algebraic Group Model", September 2020,
<<https://eprint.iacr.org/2020/1071>>.
- [PolytimeROS] "On the (in)security of ROS", July 2020,
<<https://eprint.iacr.org/2020/945>>.
- [RemoteTimingAttacks] "Remote Timing Attacks are Practical", May 2003,
<<https://crypto.stanford.edu/~dabo/papers/ssl-timing.pdf>>.
- [RFC4055] Schaad, J., Kaliski, B., and R. Housley, "Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", [RFC 4055](#), DOI 10.17487/RFC4055, June 2005,
<<https://doi.org/10.17487/RFC4055>>.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", [RFC 5280](#), DOI 10.17487/RFC5280, May 2008,
<<https://doi.org/10.17487/RFC5280>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", [RFC 8446](#), DOI 10.17487/RFC8446, August 2018,
<<https://doi.org/10.17487/RFC8446>>.
- [RSA-FDH] "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols", October 1995,
<<https://cseweb.ucsd.edu/~mihir/papers/ro.pdf>>.
- [Sch01] Schnorr, C., "Security of Blind Discrete Log Signatures against Interactive Attacks", DOI 10.1007/3-540-45600-7_1, Information and Communications Security pp. 1-12, 2001,
<https://doi.org/10.1007/3-540-45600-7_1>.

[TimingAttacks]

Kocher, P., "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems",
DOI 10.1007/3-540-68697-5_9, Advances in Cryptology -
CRYPTO '96 pp. 104-113, 1996,
<https://doi.org/10.1007/3-540-68697-5_9>.

[UProve] "U-Prove", February 2012, <<https://www.microsoft.com/en-us/research/project/u-prove/>>.

[Wagner02] Wagner, D., "A Generalized Birthday Problem",
DOI 10.1007/3-540-45708-9_19, Advances in Cryptology -
CRYPTO 2002 pp. 288-304, 2002,
<https://doi.org/10.1007/3-540-45708-9_19>.

[WM99] "Unknown key-share attacks on the station-to-station (STS) protocol", October 1999.

Appendix A. Test Vectors

This section includes test vectors for the blind signature protocol defined in this document. The following parameters are specified:

- * p, q, n, e, d: RSA private and public key parameters, each encoded as a hexadecimal string.
- * msg: Message being signed, encoded as a hexadecimal string. The hash is computed using SHA-384.
- * salt: Randomly-generated salt used when computing the signature. The length (sLen) is either 48 or 0 bytes.
- * inv: The message blinding inverse, encoded as a hexadecimal string.
- * encoded_msg: EMSA-PSS encoded message. The mask generation function is MGF1 with SHA-384.
- * blinded_msg, blind_sig: The protocol values exchanged during the computation, encoded as hexadecimal strings.
- * sig: The output message signature.

Test vector for probabilistic signatures (sLen=48):

Internet-Draft

RSA Blind Signatures

February 2022

p = e1f4d7a34802e27c7392a3cea32a262a34dc3691bd87f3f310dc756734889305
59c120fd0410194fb8a0da55bd0b81227e843fdca6692ae80e5a5d414116d4803fca
7d8c30eaaae57e44a1816ebb5c5b0606c536246c7f11985d731684150b63c9a3ad9e
41b04c0b5b27cb188a692c84696b742a80d3cd00ab891f2457443dadfeba6d6daf10
8602be26d7071803c67105a5426838e6889d77e8474b29244cefaf418e381b312048
b457d73419213063c60ee7b0d81820165864fef93523c9635c22210956e53a8d9632
2493ffc58d845368e2416e078e5bcb5d2fd68ae6acfa54f9627c42e84a9d3f277401
7e32ebca06308a12ecc290c7cd1156dcccfb2311

q = c601a9caea66dc3835827b539db9df6f6f5ae77244692780cd334a006ab353c8
06426b60718c05245650821d39445d3ab591ed10a7339f15d83fe13f6a3dfb20b945
2c6a9b42eaa62a68c970df3cadb2139f804ad8223d56108dfde30ba7d367e9b0a7a8
0c4fdb2fd9dde6661fc73fc2947569d2029f2870fc02d8325acf28c9afa19ecf962
daa7916e21afad09eb62fe9f1cf91b77dc879b7974b490d3ebd2e95426057f35d0a3
c9f45f79ac727ab81a519a8b9285932d9b2e5ccd347e59f3f32ad9ca359115e7da00
8ab7406707bd0e8e185a5ed8758b5ba266e8828f8d863ae133846304a2936ad7bc7c
9803879d2fc4a28e69291d73dbd799f8bc238385

n = aec4d69addc70b990ea66a5e70603b6fee27aafebd08f2d94cbe1250c556e047
a928d635c3f45ee9b66d1bc628a03bac9b7c3f416fe20dabea8f3d7b4bbf7f963be3
35d2328d67e6c13ee4a8f955e05a3283720d3e1f139c38e43e0338ad058a9495c533
77fc35be64d208f89b4aa721bf7f7d3fef837be2a80e0f8adf0bcd1eec5bb040443a
2b2792fdca522a7472aed74f31a1ebe1eebc1f408660a0543dfe2a850f106a617ec6
685573702eaaa21a5640a5dcaf9b74e397fa3af18a2f1b7c03ba91a6336158de420d
63188ee143866ee415735d155b7c2d854d795b7bc236cffe71542df34234221a0413
e142d8c61355cc44d45bda94204974557ac2704cd8b593f035a5724b1adf442e78c5
42cd4414fce6f1298182fb6d8e53cef1adfd2e90e1e4deec52999bdc6c29144e8d52
a125232c8c6d75c706ea3cc06841c7bda33568c63a6c03817f722b50fcf898237d78
8a4400869e44d90a3020923dc646388abcc914315215fcd1bae11b1c751fd52443aa
c8f601087d8d42737c18a3fa11ecd4131ecae017ae0a14acfc4ef85b83c19fed33cf
d1cd629da2c4c09e222b398e18d822f77bb378dea3cb360b605e5aa58b20edc29d00
0a66bd177c682a17e7eb12a63ef7c2e4183e0d898f3d6bf567ba8ae84f84f1d23bf8
b8e261c3729e2fa6d07b832e07cddd1d14f55325c6f924267957121902dc19b3b329
48bdead5

e = 010001

d = 0d43242aefe1fb2c13fbc66e20b678c4336d20b1808c558b6e62ad16a2870771
80b177e1f01b12f9c6cd6c52630257ccef26a45135a990928773f3bd2fc01a313f1d
ac97a51cec71cb1fd7efc7adffdeb05f1fb04812c924ed7f4a8269925dad88bd7dcf
bc4ef01020ebfc60cb3e04c54f981fdbd273e69a8a58b8ceb7c2d83fbc6d6f784d05
2201b88a9848186f2a45c0d2826870733e6fd9aa46983e0a6e82e35ca20a439c5ee7
b502a9062e1066493bdadf8b49eb30d9558ed85abc7afb29b3c9bc644199654a4676
681af4babcea4e6f71fe4565c9c1b85d9985b84ec1abf1a820a9bbebeee0df1398aae

2c85ab580a9f13e7743afd3108eb32100b870648fa6bc17e8abac4d3c99246b1f0ea
9f7f93a5dd5458c56d9f3f81ff2216b3c3680a13591673c43194d8e6fc93fc1e37ce
2986bd628ac48088bc723d8fbe293861ca7a9f4a73e9fa63b1b6d0074f5dea2a624c
5249ff3ad811b6255b299d6bc5451ba7477f19c5a0db690c3e6476398b1483d10314
afd38bbaf6e2fbdbcd62c3ca9797a420ca6034ec0a83360a3ee2adf4b9d4ba29731d
131b099a38d6a23cc463db754603211260e99d19affc902c915d785455aabbf608e3
ac52c19b8aa26ae042249b17b2d29669b5c859103ee53ef9bdc73ba3c6b537d5c34b
6d8f034671d7f3a8a6966cc4543df223565343154140fd7391c7e7be03e241f4ecfe

b877a051
msg = 8f3dc6fb8c4a02f4d6352edf0907822c1210a9b32f9bdda4c45a698c80023a
a6b59f8cfec5fdbb36331372ebefedae7d
salt = 051722b35f458781397c3a671a7d3bd3096503940e4c4f1aaa269d60300ce
449555cd7340100df9d46944c5356825abf
inv = 80682c48982407b489d53d1261b19ec8627d02b8cda5336750b8cee332ae26
0de57b02d72609c1e0e9f28e2040fc65b6f02d56dbd6aa9af8fde656f70495dfb723
ba01173d4707a12fddac628ca29f3e32340bd8f7ddb557cf819f6b01e445ad96f874
ba235584ee71f6581f62d4f43bf03f910f6510deb85e8ef06c7f09d9794a008be7ff
2529f0ebb69decef646387dc767b74939265fec0223aa6d84d2a8a1cc912d5ca25b4
e144ab8f6ba054b54910176d5737a2cfff011da431bd5f2a0d2d66b9e70b39f4b050e
45c0d9c16f02deda9ddf2d00f3e4b01037d7029cd49c2d46a8e1fc2c0c17520af1f4
b5e25ba396afc4cd60c494a4c426448b35b49635b337cfb08e7c22a39b256dd032c0
0adddafb51a627f99a0e1704170ac1f1912e49d9db10ec04c19c58f420212973e0cb
329524223a6aa56c7937c5dffdb5d966b6cd4cbc26f3201dd25c80960a1a111b3294
7bb78973d269fac7f5186530930ed19f68507540eed9e1bab8b00f00d8ca09b3f099
aae46180e04e3584bd7ca054df18a1504b89d1d1675d0966c4ae1407be325cdf623c
f13ff13e4a28b594d59e3eadbadf6136eee7a59d6a444c9eb4e2198e8a974f27a39e
b63af2c9af3870488b8adaad444674f512133ad80b9220e09158521614f1faadfe85
05ef57b7df6813048603f0dd04f4280177a11380fbfc861dbcbd7418d62155248dad
5fdec0991f
encoded_msg = 6e0c464d9c2f9fbc147b43570fc4f238e0d0b38870b3addcf7
a4217df912ccefc17a7f629aa850f63a063925f312d61d6437be954b45025e8282f9c
0b1131bc8ff19a8a928d859b37113db1064f92a27f64761c181c1e1f9b251ae5a2f8
a4047573b67a270584e089beadcb13e7c82337797119712e9b849ff56e04385d144d
3ca9d8d92bf78adb20b5bbeb3685f17038ec6afade3ef354429c51c687b45a7018ee
3a6966b3af15c9ba8f40e6461ba0a17ef5a799672ad882bab02b518f9da7c1a96294
5c2e9b0f02f29b31b9cdf3e633f9d9d2a22e96e1de28e25241ca7dd04147112f5789
73403e0f4fd80865965475d22294f065e17a1c4a201de93bd14223e6b1b999fd548f
2f759f52db71964528b6f15b9c2d7811f2a0a35d534b8216301c47f4f04f412cae14
2b48c4cdff78bc54df690fd43142d750c671dd8e2e938e6a440b2f825b6dbb3e19f1
d7a3c0150428a47948037c322365b7fe6fe57ac88d8f80889e9ff38177bad8c8d8d9
8db42908b389cb59692a58ce275aa15acb032ca951b3e0a3404b7f33f655b7c7d83a

2f8d1b6bbff49d5fcedf2e030e80881aa436db27a5c0dea13f32e7d460dbf01240c2
320c2bb5b3225b17145c72d61d47c8f84d1e19417ebd8ce3638a82d395cc6f7050b6
209d9283dc7b93fecc04f3f9e7f566829ac41568ef799480c733c09759aa9734e201
3d7640dc6151018ea902bc
blinded_msg = 10c166c6a711e81c46f45b18e5873cc4f494f003180dd7f115
585d871a28930259654fe28a54dab319cc5011204c8373b50a57b0fdc7a678bd74c5
23259dfe4fd5ea9f52f170e19dfa332930ad1609fc8a00902d725cfe50685c95e5b2
968c9a2828a21207fcf393d15f849769e2af34ac4259d91dfd98c3a707c509e1af55
647efaa31290ddf48e0133b798562af5eabd327270ac2fb6c594734ce339a14ea4fe
1b9a2f81c0bc230ca523bda17ff42a377266bc2778a274c0ae5ec5a8cbbe364fcf0d
2403f7ee178d77ff28b67a20c7ceec009182dbcaa9bc99b51ebbf13b7d542be33717
2c6474f2cd3561219fe0dfa3fb207cff89632091ab841cf38d8aa88af6891539f263
adb8eac6402c41b6ebd72984e43666e537f5f5fe27b2b5aa114957e9a580730308a5
f5a9c63a1eb599f093ab401d0c6003a451931b6d124180305705845060ebba6b0036
154fcef3e5e9f9e4b87e8f084542fd1dd67e7782a5585150181c01eb6d90cb958838

37384a5b91dbb606f266059ecc51b5acbaa280e45cfd2eec8cc1cdb1b7211c8e1480
5ba683f9b78824b2eb005bc8a7d7179a36c152cb87c8219e5569bba911bb32a1b923
ca83de0e03fb10fba75d85c55907dda5a2606bf918b056c3808ba496a4d955322120
40a5f44f37e1097f26dc27b98a51837daa78f23e532156296b64352669c94a8a855a
cf30533d8e0594ace7c442
blind_sig = 364f6a40dbfbc3bbb257943337eef791a0f290898a67912
83bba581d9eac90a6376a837241f5f73a78a5c6746e1306ba3adab6067c32ff69115
734ce014d354e2f259d4cbfb890244fd451a497fe6ecf9aa90d19a2d441162f7eaa7
ce3fc4e89fd4e76b7ae585be2a2c0fd6fb246b8ac8d58bcb585634e30c9168a43478
6fe5e0b74bfe8187b47ac091aa571ffea0a864cb906d0e28c77a00e8cd8f6aba4317
a8cc7bf32ce566bd1ef80c64de041728abe087bee6cadd0b7062bde5ceef308a23bd
1ccc154fd0c3a26110df6193464fc0d24ee189aea8979d722170ba945fdcce9b1b4b
63349980f3a92dc2e5418c54d38a862916926b3f9ca270a8cf40dfb9772bfbdd9a3e
0e0892369c18249211ba857f35963d0e05d8da98f1aa0c6bba58f47487b8f663e395
091275f82941830b050b260e4767ce2fa903e75ff8970c98bfb3a08d6db91ab1746c
86420ee2e909bf681cac173697135983c3594b2def673736220452fde4ddec867d40
ff42dd3da36c84e3e52508b891a00f50b4f62d112edb3b6b6cc3dbd546ba10f36b03
f06c0d82aeec3b25e127af545fac28e1613a0517a6095ad18a98ab79f68801e05c17
5e15bae21f821e80c80ab4fdec6fb34ca315e194502b8f3dcf7892b511aee45060e3
994cd15e003861bc7220a2babd7b40eda03382548a34a7110f9b1779bf3ef6011361
611e6bc5c0dc851e1509de1a
sig = 6fef8bf9bc182cd8cf7ce45c7dcf0e6f3e518ae48f06f3c670c649ac737a8b
8119a34d51641785be151a697ed7825fdfece82865123445eab03eb4bb91cecf4d69
51738495f8481151b62de869658573df4e50a95c17c31b52e154ae26a04067d5ecdc
1592c287550bb982a5bb9c30fd53a768cee6baabb3d483e9f1e2da954c7f4cf492fe
3944d2fe456c1ecaf0840369e33fb4010e6b44bb1d721840513524d8e9a3519f40d1

b81ae34fb7a31ee6b7ed641cb16c2ac999004c2191de0201457523f5a4700dd64926
7d9286f5c1d193f1454c9f868a57816bf5ff76c838a2eeb616a3fc9976f65d4371de
ecfbab29362caebdff69c635fe5a2113da4d4d8c24f0b16a0584fa05e80e607c5d9a
2f765f1f069f8d4da21f27c2a3b5c984b4ab24899bef46c6d9323df4862fe51ce300
fca40fb539c3bb7fe2dcc9409e425f2d3b95e70e9c49c5feb6ecc9d43442c33d5000
3ee936845892fb8be475647da9a080f5bc7f8a716590b3745c2209fe05b17992830c
e15f32c7b22cde755c8a2fe50bd814a0434130b807dc1b7218d4e85342d70695a5d7
f29306f25623ad1e8aa08ef71b54b8ee447b5f64e73d09bdd6c3b7ca224058d7c67c
c7551e9241688ada12d859cb7646fbd3ed8b34312f3b49d69802f0eaa11bc4211c2f
7a29cd5c01ed01a39001c5856fab36228f5ee2f2e1110811872fe7c865c42ed59029
c706195d52

Test vector for deterministic signatures (sLen=0):

p = ca9d82e9059fa3b145da850e0c451ff31093d819644ba29a3409393de2adfa1b
cd65e8669a5c5140142c1404204edbc380d4e7a5c866c06bb2427c76b9e3d16bbfc1
b1668dec219b8c59fee90b7baf557fc2feb13f2f4b30d8606d20b9928f4f588a3b34
baa659b3bd1dd590c83e90e6251b5239fbbb73b12e90534a375e3f71
q = c075694f69db6a07456e19eeace01b430f2d6cc6cd5495d569e242b6f5e8ded7
df27e6aeaa4db4e307554fb519b68279a58d9e2d25cee4b37668554eec2f2feb7924
6955a07bd526f02a6afedc7a3aff2b8953287fef2c4a02207ccb9f14e4612e9af344
7dd3401728a8957871b759b6bbf22aa0e8271b82f32dd5a2d2550197

n = 98530f850dcc894d84ecf9dec3a475bf30ec3ce4606f677ac4a6ef63f763ff
64a162ef1c991d8094b5652d0d78c126b3e97d1d77eba2f833b5be9a124e003065ec
2a3ea4fbc31bc283de1c7cd8a971eb57aa7284b082562ccde572b73702068a6143e6
dabf886538ff419874c300a85f3d9d50f0731fc6b9c92a121fefb7911f5ea92d25b1
7a4f3b2883eff34a221b5c28c488e35067a8460d8fab1c405704ebfa1ca165d69cd4
e425995a03a447f6cbba5d20d459707ab4a2c537a5dbd02801d7b19a03aaa9aec21d
1c363996c6b9fee2cab370d501c9b67e7dc4a20eb0cdc3b24be242093b5a66119b96
da0fb0ec0b1b0da0bd0b92236ece47d5c95bdca7
e = 010001
d = 6b15d18e4f8220709fe75f7226ca517ef9b7320d28dc66d54fa89a5727670f24
c7a0f1857a0c6682338946a4a298e6e90788390e137553afbbe2a4297a7edd8128d6
1b68c8e1b96b7596f0fa0406e9308e2ba64735e344edc237c97b993411b7796721ae
54d05bda1574d5af913e59e30479b373e86676cb6566f7ada0480d3ae21d50ac94c0
b41c476e566d6bcdef88eeab3042ef1016527558e794b6029cff1120596fe2104fac
928a66ad2fb1094d1ae1231abf95206cae7cd4e7aad388199d7ac1fe17e3f9174362
32cffe70e12056e02cfb9604e73cc34984bb83f7112ed197bf3a4d9f6d0c0e3c4dd8
f2d9cbe17185f1e63561b08f7d14bd36112f3ea1
msg = 5465737420766563746f7220776974682064657465726d696e697374696320
70616464696e67

```
encoded_msg = 4021ac68705782fb7587bf24ac0528853025aa4a998db7b1a503af
b5b266cbd1876710a2b0aa6e37b70fca538d42285beddd61d965c02b2162c8644587
3bdaf687a29bf6b2ab10fa22013cae53ff1c78969ef6c3eb069bfef339a5df788044
d159678e571e50fc3fa40a30fe183348453542f258c663dc9c4b372895120ad12ff8
b8ec1d37d766b2604fbf50bf9d84432a59593d21d7f379d6bf9198ea2fa90ee5abad
b27eada5d6f40a2ec45aa4bb8710042beab5c6afb4381fc57012e61b3a815800e53e
69fe2fdccb3b4ee51968c1ef6281d7e8fe08c4337bad73d99e947df834e5402378a6
6142bf032dfade7e6e2d43ae90b145055861e06eff189b63bc
inv = 6e69972553327ee6240ce0de7146aea2243927cf9f7f52c0103367df79e3ba
febfa61c2ffdc41ea397a38523654a1a806f4eebcd5fe9a2592a463f1faa26c3601f
83f29141eda488f14f7c0aa82faa025e37adbe77e02e575f72f7b9d095882923476f
2328dfaeb23b607d2f706c6c8ef6c2aee50ddb14e6d27e043e7dec8e5dede6844aa8
0b2206b6019350d37925bb8819653aa7a13bfb9cc3c95b53378f278903b5c06a10c0
b3ce0aa028e9600f7b2733f0278565f9b88e9d92e039db78300170d7bbd32ce2b89a
d8944167839880e3a2aeba05bf00edc8032a63e6279bf42a131ccc9bb95b8693764b
27665274fb673bdfb7d69b7957ee8b64a99efbeed9
blinded_msg = 5a631b41e7759a634cef04359436e358143ee2892fbeb072d1e5c
c45475ff55b6b492e13c59979f4b968994ddca3cc068084d3b176a6132039c584707
acbb9862c009fa5b63cfb7b6f6d577825c1e81ad11059cb87a524083230f906ea0a4
d9db3434d49cf9f0ea52b2425db4d319f51540e5de6cfb30b86d5e5d810a284f3478
f6259f054407c854303ec23c2e0989dd57aa002e56ab6287594c25154a1646060cb4
f6479b07f627991f7089ac0491d5841d6461166b324b3d448b2a8071de68505503fe
adf7d8182d18d8b0d3b91d77b627a5ffae68f913efbbb2fc082437f845880f94f07d
873bc0c0688f60033235bcc1701dcba83dca944b05227884e3
blind_sig = 817596a0b568088b60c29482c0178d34e0d54dc34a9375152701e4e6
d5ef76c92f4281a377d8b2f438f6af4ef9c26dd38ad2cc932f90fe45d4c0a1ba10e6
95a1c8e797aa5023f84385904e5f378df5677b8eb7312f835f9e3a097b1b7e55fece
0d00ec3f52ba26b39c91322b6404eef4e567d909195bfc0f72690805ea3f71736d7e
b51e84556c5241786f5f37bf9d2a0305bf36454d9ab8b5a9f6fe03fd4ab472b5379d
```

```
7e8ab92e803c7c15bf3d0234653e1f6d80d23c7f127bed7fba3d297b62fee51b8e71
b04d402cf291ac87460011fd222cfd27b5669d79d1e0dcc8d911c2dc6d0edcd205a9
1278cc97019cfc709ce8a50297409e66f27b1299e386a6cd
sig = 848fc8a032ea073280a7d9146ae55bb0199cd1941c10a03cce1dc38579c4e7
7e87f259e250b16a9912ce2c085cb9489846f803fd6ed09bf8605c4aa8b0ebf2c938
093e53ad025a48b97f7975255805118c33fa0f73ec204b9723acefacd8031ab3d9f7
ebeaf996eee3678c788cea96932dd723b236355c0e6864fad2fc87b00e4eda476e90
f000936b0d9fa65bf1112fc296e8aa5bb05ca7cb32dec01407e3d3ed94c1ebb0dc43
0ea59588ccc0995a6e2f1423dbe06c6f27650b23b12eb343b9e461ba532825e5e265
72fbe723b69753c178361e7a834a566ce950df55ff97d314b384b3fa8c0098d560d4
c6ba519a9b6040f908adf34f6b2d5d30c265cd0fb1
```

Authors' Addresses

Frank Denis
Fastly Inc.

Email: fd@00f.net

Frederic Jacobs
Apple Inc.

Email: frederic.jacobs@apple.com

Christopher A. Wood
Cloudflare

Email: caw@heapingbits.net