U-Prove Cryptographic Specification V1.1

Revision 5

Microsoft Corporation

Authors: Christian Paquin, Greg Zaverucha

March 2023

Summary

This document specifies the foundational U-Prove cryptographic protocols. It allows developers to create interoperable implementations of U-Prove protocol participants.

Contents

List of Figures

Change history

Version	Description
Revision 1	Initial version. Released under the OSP at http://www.microsoft.com/u-prove.
Revision 2	• Optimized issuance protocol: moved σ_z value computation from Prover to Issuer (and therefore got rid of z_i in the Issuer parameters), and move some issuance Prover computations to the precomputation phase. • Modified challenge generation. • Added the ability to generate commitments to attribute values to extend the specification. • Added the ability to present scope-exclusive pseudonyms from attribute values.
Revision 3	• Modified challenge generation in Section 2.6. • Changed hash formatting in Section 2.4.
Revision 4	• Added note about ROS attack in Section 3.8.
Revision 5	• Deprecated the Subgroup construction.

1 Introduction

This document contains the cryptographic specification for the U-Prove technology. It allows developers to create interoperable implementations of U-Prove protocol participants. See [UPTO] for an overview of the U-Prove technology, and [Brands] for background information about the cryptographic protocols. Application-specific behavior such as U-Prove token contents and encoding, and protocol extensions, must be defined in external documents.

1.1 Notation

The following notation is used throughout the document.

$a \in A$	Indicates that element a is in set A. If a is a list, then all its elements are in A.
$A \subseteq B$	Indicates that the set A is a subset of or equal to set B.
$A \cup B$	The union of the sets A and B.
$A - B$	When A and B are sets, this represents the set of elements present in A but not in B.
$\{0,1\}^*$	The set of all octet strings with a minimum length of 0 (the empty string) up to a maximum length of $2^{32} - 1$.
$\{0, 1, \dots, n, t\}$	A set of index values from 0 to n, plus a special last value labeled t. The number n could be 0, in which case the set is equal to $\{0, t\}$. In an implementation, it is safe to assume that $t = n + 1$.
$\mathbb{Z}_q$	The set of integers modulo q, i.e., $\{0, 1, \dots, q - 1\}$. In this document, q is always a large prime number.
$\mathbb{Z}_q^*$	The multiplicative subgroup of $\mathbb{Z}_q$. For a prime number q this is $\{1, \dots, q - 1\}$.
G_q	An algebraic group of prime order q. This document defines one group construction based on elliptic curves over a prime field, see Section 2.1. For uniformity, the multiplicative notation used throughout; it should be understood that ab represents the group addition of points a and b, and that a^b represents the scalar multiplication of point a by the integer b.
Ø	The null value, a zero-length octet string.
0x	Prefix of a hexadecimal value. For example, 0x39c3 represents the two octet values 39 and c3 in sequence.
$\mathbf{d}$	A Boolean value used to indicate whether or not a token is Device-protected.
$\bar{\mathbf{d}}$	Negation of the Boolean value $\mathbf{d}$.
ab	Group operation of elements a and b. For elements of $\mathbb{Z}_q$, ab means $a \times b \bmod q$; for clarity, we write $\bmod q$ explicitly in this case. For elements of G_q, the meaning of ab means the group addition of the elliptic curve points a and b.
a^{-b}	If $b = 1$, then this represents the group inverse of element a. If $b > 1$, this is equivalent to $(a^b)^{-1}$. In this document, this operation is always performed in G_q or in $\mathbb{Z}_q$.

$a := b$	Assign value b to element a.
$a\|\|b$	The binary concatenation of a and b.
$\mathcal{H}(\dots)$	Hash the input data represented by the ellipsis in a fixed order, see Section 2.2 for hash input formatting.
$\mathcal{H}_{raw}(X)$	Hash the octet string X directly without formatting (meaning without prepending its length).
$\mathcal{H}(\dots) \rightarrow \mathbb{Z}_q$	Transform the outcome of a hash operation into an element of $\mathbb{Z}_q$, see Section 2.2.
$\prod_{i \in I} a_i$	Multiply all the values a_i for which $i \in I$.
$[X]_{\mathbf{a}}$	Represents an optional operation (perform action X only if Boolean $\mathbf{a}$ is true) or an optional parameter (X is present only if Boolean $\mathbf{a}$ is true).
$\langle \dots \rangle$	A list of values to be hashed, see Section 2.2.

In protocol descriptions, the statement "Verify **X**" indicates that an error should be returned and the protocol aborted if **X** does not hold.

The key words "MUST", "MUST NOT", "SHOULD", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119].

1.2 Technology overview

The following is a brief summary of the U-Prove Technology Overview document [UPTO], which elaborates on the nature and strength of the security and privacy properties of U-Prove tokens.

A U-Prove token is a digitally signed container of attribute information of any type. It is issued to a Prover by an Issuer via an issuance protocol, and is subsequently presented by the Prover to a Verifier via a presentation protocol. The Prover can also non-interactively use U-Prove tokens to sign data and documents.

Each U-Prove token corresponds to a unique private key that the Prover generates in the issuance protocol. When using a U-Prove token, the Prover applies the token's private key to a message to create a presentation proof. This proof is a proof-of-possession of the private key as well as a digital signature of the Prover on the message. When presenting the token to a Verifier, the message can be used as a presentation challenge to prevent replay attacks. When the Prover uses the token non-interactively, the signed message can later be verified by any Verifier. The U-Prove token, the presentation proof, and the message can be kept in an audit log for later verification.

The use of a U-Prove token does not reveal its private key; this ensures that the token cannot be stolen through eavesdropping or phishing and prevents unauthorized replay by legitimate Verifiers. Arbitrarily many presentation proofs or signatures may be created with the same U-Prove token.

A U-Prove token can be made more informative at issuance time by encoding application-specific attribute information of any type into any of the following token fields:

- The token information field contains a value TI encoded by the Issuer that is always disclosed when the Prover uses the U-Prove token. A typical use of this field is to encode token metadata, such as a validity period.
- The attribute fields contain values $(\mathrm{A}_1, \ldots, \mathrm{A}_n)$ encoded by the Issuer; the Prover can selectively hide or disclose the value of each field when using the U-Prove token.
- The Prover information field contains a value PI encoded by the Prover; it is invisible to the Issuer, but is always disclosed when using the U-Prove token.

A universally unique token identifier can be computed from each U-Prove token. The Issuer cannot learn any information about this value at issuance time; as a result, it cannot be used to correlate a presented U-Prove token to its specific issuance instance. Token identifiers are particularly useful to identify repeat visitors and for token revocation.

The verification of a U-Prove token and a corresponding presentation proof requires only an authentic copy of the Issuer parameters under which the U-Prove token was issued. The Issuer parameters are generated and distributed by the Issuer.

An Issuer can issue a U-Prove token to a Prover in such a manner that the Prover cannot use the token without the assistance of a trusted Device (e.g., a smartcard, a mobile phone, or an online server). The Device can efficiently protect multiple tokens issued by any number of Issuers, and can dynamically (i.e., at token use time) enforce policies on behalf of the Issuer, Verifiers, or third parties — all without being able to compromise the Prover's privacy and without needing to interact with the Issuer.

The Prover can present a pseudonym derived from one token attribute that is unique to a particular scope. This allows Verifiers to recognize repeat visitors even if they present different U-Prove tokens, as long as they encode the same attribute.

The features included in this specification have been selected to support the majority of today's identity scenarios. There are, however, many extensions compatible with the U-Prove technology that can be implemented externally. At presentation time, a Prover can generate cryptographic commitments to encoded attributes, allowing external modules to provide additional functionality to this core specification, such as revocation, identity escrow, and advanced predicate proofs on the attributes.

In contrast to PKI certificates and other conventional authentication technologies, U-Prove tokens do not contain any unnecessary "correlation handles;" the degree to which the use of a U-Prove token can be traced to its issuance instance or linked to other uses of tokens of the same Prover is determined solely by the application-specific attribute information disclosed by the Prover. This privacy property holds even in the face of collusion between Verifiers and the Issuer.

Issuer, Prover, Verifier, and Device are basic roles. In practice, multiple roles may be performed by the same entity or a role may be split across several entities.

2 Protocol specification

The U-Prove protocols and their related artifacts are specified in this section. A conforming implementation MAY implement any combination of the Issuer, Prover, Verifier, and Device roles.

All protocol participants MUST check that all externally received mathematical elements belong to their corresponding algebraic structures prior to relying on or computing with them; failure to do so may result in critical security or privacy problems. For an element $x \in \mathbb{Z}_q$, this means verifying that $0 \leq x < q$. For an element $x \in G_q$, it is sufficient for the purpose of this specification to verify that the curve equation holds when using the elliptic curve constructions.

Test vectors from [UPCTV] can be used to validate implementations of the cryptographic protocols.

2.1 Group description

This document defines one constructions[2] for the group G_q in which it is infeasible to compute discrete logarithms.[3] It is specified by a description desc(G_q):

- **Elliptic curve construction**: The description desc(G_q) = $(p, a, b, g, q, 1)$ specifies an elliptic curve over a finite field $\mathbb{F}_p$, where p is a prime number, a and b are two field elements defining the elliptic curve, g is a base point (g_x, g_y) of prime-order q on the curve (and the generator of G_q), q is the order of the group, and 1 is the cofactor of the curve. An implementation SHOULD support the elliptic curve construction, and if it does, it MUST support the curves over prime fields defined in [UPRPP]. An element a is in G_q ($a \in G_q$) if the curve equation holds and $a^q = 1$.[4]

2.2 Hash algorithm

To prevent ambiguous interpretations of the inputs to a hash algorithm, input data MUST be encoded as follows, depending on its type:

- A byte (a.k.a. an octet): the value is encoded directly.
- The length of an octet string, the length of a list, and the index of an attribute: the binary value is conditionally zero-extended to a length of 32 bits. The four bytes forming the extended value are then encoded, leading with the most-significant byte (e.g., the value 11588062 is encoded as 0x00b0d1de). Such values are therefore in the range $\{0, \ldots, 2^{32} - 1\}$; larger values MUST be rejected.
- An octet string: the length of the string is encoded followed by the contents of the string (e.g., the string 0x01fe is encoded as 0x0000000201fe).
- An element of $\mathbb{Z}_q$, an element of G_q, and the values p, a, b and q in desc(G_q) for an elliptic curve construction: the binary value is conditionally zero-extended to make its length a multiple of 8 bits (the value 0 is zero-extended to a full 8 bits). The bytes forming the extended value are then encoded as an octet string, leading with the most-significant byte (e.g., the number 254666256150 is encoded as 0x000000053b4b4aaf16).

[2] See Section 2.2.2 of [Brands] for more information on other group constructions.
[3] The security properties of U-Prove tokens rely on this assumption. The privacy properties, on the other hand, hold unconditionally, limited only by the quality of the Prover-generated random numbers.
[4] Only curves with prime order are allowed by this specification. Since the cofactor is 1 for curves of prime order, all curve points are part of the group, and therefore checking that the curve equation holds is enough to verify that a point is part of the group.

- A list (delimited with ⟨…⟩): the length of the list is encoded followed by the recursive encoding of the list elements, in order.
- The null value (Ø): a zero-length octet string is encoded, yielding the sequence 0x00000000.
- A point $e = (e_x, e_y)$ on an elliptic curve (all elements of G_q when using the elliptic curve construction): the point is converted to an octet string following the procedure described in Section 2.3.3 of [SEC1], without using point compression.

To transform the outcome of a hash operation into an element of $\mathbb{Z}_q$, when $\mathcal{H}(\dots) \rightarrow \mathbb{Z}_q$ is used, the hash digest bytes are interpreted as an unsigned integer in big-endian byte-order modulo q.

The security of the protocols critically depends on the choice of the hash function, see section 3.2.

2.3 Basic primitives

In this section, we provide the mathematical specification of all artifacts related to U-Prove tokens.

2.3.1 Issuer parameters

An instance of the Issuer parameters is of the form

$$\mathrm{UID_P}, desc(G_q), \mathrm{UID}_{\mathcal{H}}, (g_0, g_1, \dots, g_n, g_t), (e_1, \dots, e_n), S$$

where:

- $\mathrm{UID_P}$ is an octet string that holds an application-specific unique identifier for the Issuer parameters, the value of which MUST be unique across the application realm.[5]
- desc(G_q) specifies a group G_q of prime order q in which it is infeasible to compute discrete logarithms. One construction is supported in this specification: the elliptic curve construction, see Section 2.1.
- $\mathrm{UID}_{\mathcal{H}}$ is an identifier of a cryptographically secure hash algorithm, see Section 3.2 for details on the security requirements for the hash algorithm.
- $(g_0, g_1, \dots, g_n, g_t)$ is the Issuer's public key. To generate g_0, the Issuer generates a private key y_0 at random from $\mathbb{Z}_q^*$ and computes $g_0 := g^{y_0}$. The private key y_0 needs to be protected appropriately. The remaining g_i values MUST be random generators of G_q. An implementation SHOULD support the pre-generated $(g_1, \dots, g_{50})$ values defined in [UPRPP] for the group constructions it supports.
- $(e_1, \dots, e_n)$ is a list of byte values indicating whether or not the attribute values $(\mathrm{A}_1, \dots, \mathrm{A}_n)$ are hashed when computing a U-Prove token public key h, see Section 2.3.5.
- S is an octet string that holds an application-specific specification for the Issuer parameters and the U-Prove tokens issued using them.

The application-specific value n specifies the number of attributes encoded into each U-Prove token that will be issued using these Issuer parameters. If no attributes are to be encoded, then the Issuer's public key becomes (g_0, g_t), and the list of e_i values is empty.

The same group description $desc(G_q)$, hash algorithm identifier $\mathrm{UID}_{\mathcal{H}}$, and the public values $(g_1, \dots, g_n, g_t)$ MAY be reused in different instances of Issuer parameters, even by different Issuers.[6] Each Issuer SHOULD, however, generate its own private key y_0 and its own g_0.

[5] For example, an application MAY choose to make the $\mathrm{UID_P}$ the digest of the other fields using $\mathrm{UID}_{\mathcal{H}}$ as a hash function.

To be assured of the privacy properties of U-Prove tokens,[7] Provers and Verifiers SHOULD verify any Issuer parameters they rely on.[8] This verification, specified in Figure 1, involves verifying that Issuer public key $(g_0, g_1, \dots, g_n, g_t)$ are well-formed. Verification of the group G_q and generators $(g_1, \dots, g_n, g_t)$ MAY be skipped when using the recommended values defined in [UPRPP].

Input
Group description: desc(G_q)
Public generators: $(g_0, g_1, \dots, g_n, g_t) \in G_q$

Verify public key elements
For each $i \in \{0,1, \dots, n, t\}$
Verify that $g_i \in G_q$ and that $g_i \neq 1$

Figure 1: Verify the Issuer parameters

In addition, Provers and Verifiers SHOULD ensure that the application-specific specification S complies with their own policy. The distribution and trust management of the Issuer parameters are outside the scope of this document, see Section 3.6.

2.3.2 Device parameters

When U-Prove tokens are Device-protected, protocol participants require additional setup data:

- $g_d \in G_q$ is the Device generator. It MUST be a generator of G_q. This value is either generated by the Issuer when creating the Issuer parameters or specified in a profile and shared by many Issuers and Devices. An implementation SHOULD support the pre-generated g_d values defined in [UPRPP].
- $x_d \in \mathbb{Z}_q^*$ is the Device private key. It MUST only be known to the Device. It can either be generated at random, or derived from a secret using a strong derivation method (such as secure hash function). The latter method is useful to generate a unique key pair per Issuer, to provide "Device unlinkability".[10] The Device MUST NOT, however, apply its private key on values from different groups G_q.
- $h_d = g_d^{x_d} \in G_q$ is the Device public key. It is computed by the Device and made available to the Prover and the Issuer in the issuance protocol. The Issuer and Prover MUST check that the Device public key is a valid element of G_q, see Section 2.1.

The distribution and trust management of the Device parameters are outside the scope of this document, see Section 3.6.

2.3.3 U-Prove token

A U-Prove token is of the form

---

[6] In case Issuers don't trust each other, they should verify that the values have been generated "verifiably at random" to ensure no Issuer knows any relative discrete logarithm of a generator with respect to any other generator. Using the values in [UPRPP] is recommended.
[7] See Section 4 of [UPTO].
[8] The privacy properties of U-Prove tokens are guaranteed only if the Issuer parameters are valid; in particular, an Issuer that can get away with an invalid group might be able to trace or link issued U-Prove tokens with significant success probability.
[10] The Device's public key acts as a unique identifier for the Prover if reused across multiple Issuers (although this is value is never seen by Verifiers). Using a per-Issuer value prevents Issuers from determining that they are issuing tokens to the same Prover based on the Device's public key.

$$\mathrm{UID_P}, h, TI, PI, \sigma'_z, \sigma'_c, \sigma'_r, \mathbf{d}$$

where:

- $\mathrm{UID_P}$ is the unique identifier of the Issuer parameters under which the U-Prove token was issued, see Section 2.3.1.
- $h \in G_q$ is the public key of the U-Prove token, see Section 2.3.5.
- $TI \in \{0,1\}^*$ is the value of the token information field. The token information field is used to encode token-specific information that is always disclosed to Verifiers, such as token usage restrictions, a validity period (see Section 3.3), or token metadata.
- $PI \in \{0,1\}^*$ is the value of the Prover information field. The Prover information field is used to encode Prover-asserted information hidden from the Issuer, such as contact information, an encryption key, or a Verifier-supplied nonce to guarantee freshness of the U-Prove token (see Section 3.3). PI is always revealed during token presentation.
- $\sigma'_z \in G_q$ and $(\sigma'_c, \sigma'_r) \in \mathbb{Z}_q$ form the Issuer's signature on all the other token contents, see Section 2.3.6.
- Boolean $\mathbf{d}$ indicates if the token is protected by a Device (**true**) or not (**false**).

For the cryptographic specification of the issuance protocol in which these elements are processed or generated, see Section 2.4.

2.3.4 U-Prove token private key

The private key of the U-Prove token, corresponding to its public key h, is the value $\alpha^{-1} \in \mathbb{Z}_q^*$, the modular multiplicative inverse of a secret value $\alpha \in \mathbb{Z}_q^*$ that the Prover generates in the issuance protocol; see Section 2.4.

The application-specific attribute values $(\mathrm{A}_1, \ldots, \mathrm{A}_n) \in \{0,1\}^*$ encoded into the U-Prove token are not part of the private key *per se*; they are typically shared among multiple U-Prove tokens and are selectively disclosed to Verifiers. However, since they are all needed to use the U-Prove token and may contain confidential user information, they SHOULD be kept secret and be handled with similar care as the private key.

2.3.5 U-Prove token public key

The public key $h \in G_q$ of a U-Prove token is of the form

$$h = (g_0 g_1^{x_1} \ldots g_n^{x_n} g_t^{x_t} \left[g_d^{x_d}\right]_{\mathbf{d}})^{\alpha}$$

where:

- $(g_0, g_1, \ldots, g_n, g_t) \in G_q$ is the Issuer's public key, taken from the Issuer parameters under which the U-Prove token was issued; see Section 2.3.1.
- $\alpha \in \mathbb{Z}_q^*$ is a secret value generated by the Prover to randomize the public key h, see Section 2.4.
- $x_t \in \mathbb{Z}_q$ is computed by hashing the issuance protocol version 0x01, a digest of the Issuer parameters, and the token information field value TI; see Figure 2.
- The optional $g_d \in G_q$, present if the token is protected by a Device ($\mathbf{d} = \mathbf{true}$), is the public Device generator. $x_d \in \mathbb{Z}_q$ is the Device private key, known only by the Device. The Device public key $h_d = g_d^{x_d}$ allows the Prover to compute the token public key h. See Section 2.3.2.
- $x_i \in \mathbb{Z}_q$ $(1 \leq i \leq n)$ is obtained from the corresponding attribute value A_i either by hashing it (if the Issuer parameters value e_i is equal to 0x01) or by encoding it directly (if e_i is equal to 0x00). In this latter case, the value A_i is interpreted as the binary encoding of an unsigned integer in big-endian

byte-order, which must be smaller than q to be a valid element of $\mathbb{Z}_q$.[11] For efficiency reasons, x_i is set to zero if the value A_i is null (Ø) and e_i is equal to 0x01. See Figure 3.

ComputeXt( )

Input

Issuer parameter fields: $\mathrm{UID_P}, \mathrm{desc}(G_q), \mathrm{UID}_{\mathcal{H}}, (g_0, g_1, \ldots, g_n, g_t), (e_1, \ldots, e_n), S$
Token information field: $TI \in \{0,1\}^*$
Device-protected Boolean: $\mathbf{d}$
[Device generator: g_d]$_{\mathbf{d}}$

Computation

$P := \mathcal{H}(\mathrm{UID_P}, \mathrm{desc}(G_q), \langle g_0, g_1, \ldots, g_n, g_t, [g_d]_{\mathbf{d}} \rangle, \langle e_1, \ldots, e_n \rangle, S)$
Return $x_t := \mathcal{H}(0\mathrm{x}01, P, TI) \rightarrow \mathbb{Z}_q$

Figure 2: Function ComputeXt

ComputeXi( )

Input

Issuer parameter fields: q, $\mathrm{UID}_{\mathcal{H}}, e_i$
Attribute value: $\mathrm{A}_i \in \{0,1\}^*$

Computation

If e_i = 0x01
  If $\mathrm{A}_i = \emptyset$ then $x_i := 0$
  Else $x_i := \mathcal{H}(\mathrm{A}_i) \rightarrow \mathbb{Z}_q$
Else if e_i = 0x00
  Verify that $0 \leq \mathrm{A}_i < q$
  $x_i := \mathrm{A}_i$
Else return an error
Return x_i

Figure 3: Function ComputeXi

The public key of a U-Prove token is computed by the Prover in the issuance protocol, see Section 2.4. It is never seen by the Issuer and therefore it cannot be used to correlate a presented U-Prove token to its specific issuance instance.

2.3.6 Issuer's signature

The Issuer's signature in the U-Prove token is composed of the values $\sigma_z' \in G_q$ and $(\sigma_c', \sigma_r') \in \mathbb{Z}_q$. Its verification is specified in Figure 4.

[11] Not hashing the value of an attribute may improve computational performance since the efficiency of computing a modular exponentiation depends on the exponent size: the hash digest is typically a large number in $\mathbb{Z}_q$ even if the preimage is small. Not hashing the value is also preferred when encoding confidential information serving as a lending disincentive; see Section 3.5.

VerifyTokenSignature( )

Input

Issuer parameter fields: $\text{desc}(G_q), \text{UID}_{\mathcal{H}}, g_0$

U-Prove token fields: $h, PI, \sigma'_z, \sigma'_c, \sigma'_r$

Verification

Verify that $h \neq 1$

Verify that $\sigma'_c = \mathcal{H}(h, PI, \sigma'_z, g^{\sigma'_r} g_0^{-\sigma'_c}, h^{\sigma'_r} (\sigma'_z)^{-\sigma'_c}) \rightarrow \mathbb{Z}_q$

Figure 4: Function `VerifyTokenSignature`

The Issuer never sees the value of its signature and therefore it cannot be used to correlate a presented U-Prove token to its specific issuance instance.

2.3.7 Token identifier

The token identifier $\text{UID}_{\mathcal{T}}$ of a U-Prove token is computed by hashing the U-Prove token's public key and the Issuer's signature; see Figure 5.

ComputeTokenID( )

Input

Issuer parameters field: $\text{UID}_{\mathcal{H}}$

U-Prove token fields: $h, \sigma'_z, \sigma'_c, \sigma'_r$

Computation

Return $\text{UID}_{\mathcal{T}} := \mathcal{H}(h, \sigma'_z, \sigma'_c, \sigma'_r)$

Figure 5: Function `ComputeTokenID`

Owing to the security properties of the hash algorithm (see Section 3.2) and the fact that its inputs are generated mutually at random in the issuance protocol by the Prover and the Issuer (see Section 2.4), the token identifier of a U-Prove token is a unique random number (assuming the Issuer's signature is correct). To be convinced of the uniqueness of the token identifier, participants SHOULD only accept Issuer parameters that specify hash algorithms they trust to be secure.

2.4 Creating verifiable generators

The protocols require that some group elements be generated verifiably without known relationships to other group elements. To achieve this, the derivation methods specified in the following sections are used.

[UPRPP] defines pre-generated groups and associated generators that can be reused by many Issuers. The generators were created with the following procedure and this can be verified using this derivation method.

2.4.1 Elliptic curve construction

Figure 6 describes the method to generate a random group element when the elliptic curve construction is used. The method was adapted from the procedure described in case I of section D.3.1 of [ANSI X9.62], with

modified steps 1 and 5.[13] Note that two y values can be returned when computing the square root of α, so to insure interoperability between implementations, the smaller square root value is used. [14]

ComputeVerifiablyRandomElement( )

Input

Issuer parameters fields: $\text{UID}_{\mathcal{H}}, p, a, b$
$context \in \{0,1\}^*$
$index \in \{0, \ldots, 255\}$

Computation

byte $counter := 0$
byte numIteration := ⌈ bit length of p / output bit length of $\mathcal{H}$ ⌉
While $y \neq \emptyset$
  For $0 \leq$ byte $iteration < numIteration$
    $x_{iteration} := \mathcal{H}_{raw}(context$ || index || counter || iteration)
  $x := x_0$ || ... || $x_{numIteration-1}$ mod p
  $z := x^3 + ax + b$ mod p
  If $z = 0$ then $y := 0$
  Else
    $y := \sqrt{z}$ mod p
    Verify that $counter < 255$
    If $y = \emptyset$ then $counter := counter + 1$
Return point $(x, \min(y, p - y))$

Figure 6: Function `ComputeVerifiablyRandomElement` (elliptic curve construction)

2.5 Issuing U-Prove tokens

The protocol for issuing a single U-Prove token is specified in Figure 7. Some clarifications are in order:

- How the Issuer and Prover agree on the contents of the issued U-Prove token is outside the scope of this specification.
- How the Device public key is provided to the Issuer and Prover is outside the scope of this specification. If the Issuer gets this value from the Prover, then it SHOULD be authenticated to make sure the Prover has access to the Device, and that the key corresponds to a valid Device.
- The Prover's and the Issuer's precomputation steps MAY be performed as soon as the protocol inputs are known.
- Multiple U-Prove tokens generated using identical common inputs MAY be issued in parallel, in which case the three exchanged protocol messages contain values for all the parallel issuance instances. When doing so, the Prover MAY specify different values for the Prover information field. The Prover SHOULD use fresh random numbers for each issuance instance. The computation of the Prover's γ and σ_z, and the Issuer's γ can be shared among all parallel protocol executions. To improve performance, the token signature validation can be batched using a probabilistic[15] method: suppose we have k tokens, and let $\alpha_i, \sigma'_{ri}, \sigma'_{ci}, \sigma'_{ai}, \sigma'_{bi}$ be values from the i^{th} token, and γ, g, g_0, σ_z be values that are shared across all the tokens.

[13] The modification enables the generation of a verifiably random x-coordinate, and iteration over a counter if the procedure fails.
[14] An algorithm to compute the square root modulo p can be found in Annex D.1.4 of [ANSI X9.62].
[15] The probability that the Prover accepts if one of the tokens is invalid is $2^{-\ell}$, for a parameter ℓ.

1. Choose n random integers $s_1, \ldots, s_k$ from $\{1, \ldots, 2^\ell\}$, for ℓ such that $2^\ell < q$
2. Compute $\rho_{\alpha r} \coloneqq \sum_1^k s_i \alpha_i \sigma_{ri}{}' \bmod q$
3. $\rho_r \coloneqq \sum_1^k s_i \sigma_{ri}{}' \bmod q$
4. $\rho_{\alpha c} \coloneqq \sum_1^k s_i \alpha_i \sigma_{ci}{}' \bmod q$
5. $\rho_c \coloneqq \sum_1^k s_i \sigma_{ci}{}' \bmod q$
6. Verify that $\prod_1^k (\sigma'_{ai} \sigma'_{bi})^{s_i} = g^{\rho_r} \gamma^{\rho_{\alpha r}} g_0^{-\rho_c} \sigma_z^{-\rho_{\alpha c}}$

- If the Prover and the Issuer use different protocol inputs, or if one of them deviates from the issuance protocol, the protocol will result in a U-Prove token with an invalid Issuer's signature. Signature verification by the Prover at the end of the issuance protocol ensures that issued U-Prove tokens are valid. In the interest of improved performance, however, the Prover MAY skip verification of the Issuer's response σ_r and instead verify the Issuer's signature before using the U-Prove token (see Figure 4).
- The issuance protocol cannot be implemented in a stateless manner. Concretely, the Issuer MUST NOT export the value w (even in encrypted form) to the Prover when sending the first issuance message since doing so would allow an attacker to compute its private key.
- Upon completion of the protocol, the Issuer MUST delete the value w; its leakage would allow the computation of its private key. Similarly, the value w MUST NOT be reused. The Prover SHOULD delete the values , β_1, β_2, t_a, and t_b.
- Apart from the token information value TI and any Prover-disclosed values A_i, there is nothing in a U-Prove token that can be used to trace its use to its issuance or to link it to other U-Prove tokens of the same Prover. If an application relies on the untraceability and unlinkability of U-Prove tokens, precautions should be put in place to prevent information leakage through these fields. Specifically, the possible field values SHOULD be clearly defined in the Issuer parameters' specification S, and the Prover SHOULD have a means to inspect and boycott inappropriate values.
- The Issuer MUST make sure Device public key h_d is well-formed and is associated with the Prover.
- This protocol is based on the techniques described in Section 4.5.2 of [Brands].

Common input
Issuer parameters: $IP = \mathrm{UID}_\mathrm{P}, \mathrm{desc}(G_q), \mathrm{UID}_\mathcal{H}, (g_0, g_1, \dots, g_n, g_t), (e_1, \dots, e_n), S$
Application-specific attribute information: $(\mathrm{A}_1, \dots, \mathrm{A}_n)$, $TI \in \{0,1\}^*$
Device-protected Boolean: $\mathbf{d}$
[Device parameters: $g_d, h_d]_\mathbf{d}$

Prover		Issuer
Input $x_t := \mathrm{ComputeXt}(IP, TI, \mathbf{d}, [g_d]_\mathbf{d})$ For each $i \in \{1, \dots, n\}$, $x_i := \mathrm{ComputeXi}(IP, \mathrm{A}_i)$ $\gamma := g_0 g_1^{x_1} \dots g_n^{x_n} g_t^{x_t} [h_d]_\mathbf{d}$ Prover information field value: $PI \in \{0,1\}^*$		**Input** $x_t := \mathrm{ComputeXt}(IP, TI, \mathbf{d}, [g_d]_\mathbf{d})$ For each $i \in \{1, \dots, n\}$, $x_i := \mathrm{ComputeXi}(IP, \mathrm{A}_i)$ $\gamma := g_0 g_1^{x_1} \dots g_n^{x_n} g_t^{x_t} [h_d]_\mathbf{d}$ Private key: $y_0 \in \mathbb{Z}_q$ $\sigma_z := \gamma^{y_0}$
Precomputation Generate α at random from $\mathbb{Z}_q^*$ Generate β_1 and β_2 at random from $\mathbb{Z}_\mathrm{q}$ $h := \gamma^\alpha$ $t_1 = g_0^{\beta_1} g^{\beta_2}$ $t_2 = h^{\beta_2}$ Compute $\alpha^{-1} \bmod q$		**Precomputation** Generate w at random from $\mathbb{Z}_q$ $\sigma_a := g^w$ $\sigma_b := \gamma^w$
	$\sigma_z, \ \sigma_a, \sigma_b$ ←	**First message**
Second message $\sigma_z' := \sigma_z^\alpha$ $\sigma_a' := t_1 \sigma_a$ $\sigma_b' := (\sigma_z')^{\beta_1} t_2 \sigma_b^\alpha$ $\sigma_c' := \mathcal{H}(h, PI, \sigma_z', \sigma_a', \sigma_b') \to \mathbb{Z}_q$ $\sigma_c := \sigma_c' + \beta_1 \bmod q$		
	σ_c →	**Third message** $\sigma_r := \sigma_c y_0 + w \bmod q$ Delete w
	σ_r ←	
U-Prove token generation $\sigma_r' := \sigma_r + \beta_2 \bmod q$ Verify that $\sigma_a' \sigma_b' = (gh)^{\sigma_r'} (g_0 \sigma_z')^{-\sigma_c'}$ (RECOMMENDED) Delete $\beta_0, \beta_1, \beta_2, t_1, t_2$ Return U-Prove token: $\mathcal{T} = \mathrm{UID}_\mathrm{P}, h, TI, PI, \sigma_z', \sigma_c', \sigma_r', \mathbf{d}$ Private key: α^{-1}		

Figure 7: Issuance protocol

2.6 Presenting U-Prove tokens

To present an issued U-Prove token $\mathcal{T}$, the Prover transmits $\mathcal{T}$ itself, the subset of the attribute values it expressly wants to disclose to the Verifier, and a presentation proof generated by applying its token private key to a message and the non-disclosed attribute values. Figure 8 specifies the generation by the Prover of a

presentation proof. Verification by the Verifier of the U-Prove token and the presentation proof is specified in Figure 9. Some clarifications are in order:

- How the Prover, the Verifier and the optional Device agree on the protocol inputs and, more generally, the requirements to be met by the presented U-Prove token is outside the scope of this specification.
- How the Prover invokes the Device if the token is Device-protected is outside the scope of this specification.
- The presentation proof serves two security purposes: it proves the integrity and source authenticity of the Prover-disclosed attribute values, and establishes that the Prover owns the private key associated with the presented U-Prove token. The latter serves to prevent unauthorized token replay.
- The presentation proof doubles as a digital signature on the application-specific message (m, m_d). The Prover can non-interactively use the presentation proof generation to create a digital signature on arbitrary data; a Verifier can at any later time verify the signature by performing the presentation proof verification.
- The message (m, m_d)MAY be specified by either the Prover or the Verifier, depending on the application. If replay protection is required, then the message SHOULD contain a unique Verifier identifier and fresh information that cannot be predicted or anticipated (such as a cryptographic nonce); the Verifier SHOULD verify that it is in its scope and that the presentation proof is fresh.[16] The m value of the message is only visible to the Prover and Verifier, and is normally used to encode the application-specific data. If a Device or an extension is used that requires direct access to some protocol data (for example, a policy for the Device), then m_d MAY be used to encode this data; otherwise it can be left null.
- The Prover MUST NOT reuse the w_i random values when presenting the same U-Prove token. Similarly, the Device MUST NOT reuse the value w'_d.
- A Verifier MAY skip the verification of the Issuer's signature in a previously verified U-Prove token (i.e., in Figure 9, the call to `VerifyTokenSignature` function MAY be skipped). This is useful when verifying multiple presentation proofs created using the same U-Prove token, in parallel or over time.
- Both the Prover and the Verifier MAY independently archive (e.g., in an audit log) the protocol inputs specified in Figure 9; this data constitutes an unforgeable proof of the token presentation. It MAY later be verified by any party following the procedure specified in Figure 9.
- A Verifier MAY request the presentation of a scope-exclusive pseudonym, i.e., a pseudonym derived from a token attribute unique to a specific scope. An Issuer knowing the value of the attribute used to derive a pseudonym will be able to trace its usage, it is therefore RECOMMENDED to use a Device pseudonym if this is undesirable. The pseudonym generator g_s can be precomputed by Provers and Verifiers if the same scope is used in multiple token presentations.
- The Prover can cryptographically commit to the values encoded in some attributes to allow an external module to extend the functionality of this core specification. The commitments $\{(\tilde{c}_i, \tilde{a}_i, \tilde{r}_i)\}_{i \in C}$ are needed by the Verifier and are part of the presentation proof, while the $\{\tilde{o}_i\}_{i \in C}$ values are secret and only used by the Prover's extension modules. If the proof verifies, the Verifier has assurance that $\tilde{c}_i$ is a commitment to the attribute value encoded in the token $\mathcal{T}$.
- This protocol is based on the techniques described in Section 3.3.1 and 6.3.1 of [Brands].

[16] This can be achieved by keeping the received presentation proof's value a indexed by the timestamp for a short period of time, and making sure that the same value is not received again.

Device	Prover
Input	**Input**
Issuer parameters fields: $\text{desc}(G_q)$,$\text{UID}_{\mathcal{H}}$	Issuer parameter fields: $\text{UID}_\text{P}, \text{desc}(G_q), \text{UID}_{\mathcal{H}}, (g_0, g_1, \dots, g_n, g_t), (e_1, \dots, e_n), S$
Device generator: g_d	Ordered indices of disclosed attributes: $D \subset \{1, \dots, n\}$
Private key: x_d	Ordered indices of undisclosed attributes: $U = \{1, \dots, n\} - D$
	Ordered indices of committed attributes: $C \subset U$
	Pseudonym attribute index: $p \in U \cup \{d\}$
	Pseudonym scope: $s \in \{0,1\}^*$
	Messages: $m, m_d \in \{0,1\}^*$
	U-Prove token: $\mathcal{T} = \text{UID}_\text{P}, h, TI, PI, \sigma'_z, \sigma'_c, \sigma'_r, \mathbf{d}$
	Private key: $\alpha^{-1} \in \mathbb{Z}_q$
	Attribute values: $(\text{A}_1, \dots, \text{A}_n) \in \{0,1\}^*$
	[Device generator: $g_d]_\mathbf{d}$
	Presentation proof generation
	For each $i \in \{1, \dots, n\}$, $x_i \coloneqq \text{ComputeXi}(IP, \text{A}_i)$
Device commitment	Generate w_0 [and $w_d]_\mathbf{d}$ at random from $\mathbb{Z}_q$
Generate w'_d at random from $\mathbb{Z}_q$	For each $i \in U$, generate w_i at random from $\mathbb{Z}_q$
$a_d \coloneqq g_d^{w'_d}$	$\left[\xleftarrow{\quad s \quad}\right]_{\mathbf{d} \text{ and } p=d}$
If $s \neq \emptyset$	
g_s:= $\text{GenerateScopeElement}(\text{desc}(G_q), s)$	
$a'_p \coloneqq g_s^{w'_d}$	
$P_s \coloneqq g_s^{x_d}$	
	$\left[\xrightarrow{\quad a_d, [a'_p,\ P_s]_{p=d} \quad}\right]_\mathbf{d}$
	$a \coloneqq \mathcal{H}(h^{w_0}(\prod_{i \in U} g_i^{w_i})\ [g_d^{w_d} a_d]_\mathbf{d})$
	If $p \neq \emptyset$ and $s \neq \emptyset$,
	$g_s \coloneqq \text{GenerateScopeElement}(\text{desc}(G_q), s)$
	$a_p \coloneqq \mathcal{H}(g_s^{w_p}[a'_p]_{p=d})$
	$[P_s \coloneqq g_s^{x_p}]_{p \neq d}$
	Else $a_p \coloneqq \emptyset$ and $P_s \coloneqq \emptyset$
	For each $i \in C$
	Generate $\tilde{o}_i, \tilde{w}_i$ at random from $\mathbb{Z}_q$
	$\tilde{c}_i \coloneqq g^{x_i} g_1^{\tilde{o}_i}$
	$\tilde{a}_i \coloneqq \mathcal{H}(g^{w_i} g_1^{\tilde{w}_i})$
	$\text{UID}_\mathcal{T} \coloneqq \text{ComputeTokenID}(IP, \mathcal{T})$
	If $p = d$ then $p' \coloneqq 0$ else $p' \coloneqq p$
	$c_p \coloneqq \mathcal{H}(\text{UID}_\mathcal{T}, a, \langle D \rangle, \langle \{x_i\}_{i \in D} \rangle, \langle C \rangle, \langle \{\tilde{c}_i\}_{i \in C} \rangle, \langle \{\tilde{a}_i\}_{i \in C} \rangle, p', a_p, P_s, m)$
	$c \coloneqq \mathcal{H}(\langle c_p, m_d \rangle) \to \mathbb{Z}_q$
	$r_0 \coloneqq c\alpha^{-1} + w_0 \bmod q$
	For each $i \in U$, $r_i \coloneqq -cx_i + w_i \bmod q$
	$\left[\xleftarrow{\quad c_p,\ m_d \quad}\right]_\mathbf{d}$

(continued on next page)

Device response computation

Process m_d

$c \coloneqq \mathcal{H}(\langle c_p, m_d \rangle) \rightarrow \mathbb{Z}_q$

$r'_d \coloneqq -cx_d + w'_d \bmod q$

Delete w'_d

$\left[\xrightarrow{\quad r'_d \quad} \right]_{\mathbf{d}}$

$[r_d \coloneqq r'_d + w_d \bmod q]_{\mathbf{d}}$

For each $i \in C$, $\tilde{r}_i \coloneqq -c\tilde{o}_i + \tilde{w}_i \bmod q$

Delete $w_0, \{w_i\}_{i \in U}, [w_d]_{\mathbf{d}}, \{\tilde{w}_i\}_{i \in C}$

Return

Presentation proof: $\{A_i\}_{i \in D}, a, (a_p, P_s), r_0, \{r_i\}_{i \in U}, [r_d]_{\mathbf{d}}, \{(\tilde{c}_i, \tilde{a}_i, \tilde{r}_i)\}_{i \in C}$

Secret commitment values: $\{\tilde{o}_i\}_{i \in C}$

Figure 8: Subset presentation proof generation

Input

Issuer parameter fields: $IP = \text{UID}_\text{P}, \text{desc}(G_q), \text{UID}_\mathcal{H}, (g_0, g_1, \ldots, g_n, g_t), (e_1, \ldots, e_n), S$

Ordered indices of disclosed attributes: $D \subset \{1, \ldots, n\}$

Ordered indices of undisclosed attributes: $U = \{1, \ldots, n\} - D$

Ordered indices of committed attributes: $C \subset U$

U-Prove token: $\mathcal{T} = \text{UID}_\text{P}, h, TI, PI, \sigma'_z, \sigma'_c, \sigma'_r, \mathbf{d}$

Pseudonym attribute index: $p \in U \cup \{d\}$

Pseudonym scope: $s \in \{0,1\}^*$

Messages: $m, m_d \in \{0,1\}^*$

Presentation proof: $\{\text{A}_i\}_{i \in D}, a, (a_p, P_s), r_0, \{r_i\}_{i \in U}, [r_d]_{\mathbf{d}}, \{(\tilde{c}_i, \tilde{a}_i, \tilde{r}_i)\}_{i \in C}$

[Device generator: $g_d]_{\mathbf{d}}$

U-Prove token verification

VerifyTokenSignature$(IP, \mathcal{T})$

Presentation proof verification

$x_t \coloneqq \text{ComputeXt}(IP, TI, \mathbf{d}, [g_d]_{\mathbf{d}})$

For each $i \in D$, $x_i \coloneqq \text{ComputeXi}(IP, \text{A}_i)$

$\text{UID}_\mathcal{T} \coloneqq \text{ComputeTokenID}(IP, \mathcal{T})$

If $p = d$ then $p' \coloneqq 0$ else $p' \coloneqq p$

$c_p \coloneqq \mathcal{H}(\text{UID}_\mathcal{T}, a, \langle D \rangle, \langle \{x_i\}_{i \in D} \rangle, \langle C \rangle, \langle \{\tilde{c}_i\}_{i \in C} \rangle, \langle \{\tilde{a}_i\}_{i \in C} \rangle, p', a_p, P_s, m)$

$c \coloneqq \mathcal{H}(\langle c_p, m_d \rangle) \rightarrow \mathbb{Z}_q$

Verify that $a = \mathcal{H}\left(\left(g_0 g_t^{x_t} \prod_{i \in D} g_i^{x_i} \right)^{-c} h^{r_0} \left(\prod_{i \in U} g_i^{r_i} \right) \left[g_d^{r_d} \right]_{\mathbf{d}} \right)$

If $a_p \neq \emptyset$ and $P_s \neq \emptyset$

$g_s \coloneqq \text{GenerateScopeElement}(\text{desc}(G_q), \text{UID}_\mathcal{H}, s)$

Verify that $a_p = \mathcal{H}(P_s^c g_s^{r_p})$

For each $i \in C$, verify that $\tilde{a}_i = \mathcal{H}(\tilde{c}_i^c g^{r_i} g_1^{\tilde{r}_i})$

Figure 9: Subset presentation proof verification

GenerateScopeElement( )

Input

Hash identifier: $\mathrm{UID}_{\mathcal{H}}$
Group description: $\mathrm{desc}(G_q)$
Scope string: $s \in \{0,1\}^*$

Compute scope generator

Return $g_s := \mathrm{ComputeVerifiablyRandomElement}\left(\mathrm{UID}_{\mathcal{H}}, \mathrm{desc}\left(G_q\right), s, 0\right)$

Figure 10: Generate a group element given a Verifier scope

3 Security considerations

3.1 Key sizes

The performance and the security properties (but not the privacy properties) of the U-Prove protocols depend on the sizes of elements in G_q and in $\mathbb{Z}_q$.

In contrast to conventional digital certificates, the size of the public key of a U-Prove token cannot be picked independently from the size of the Issuer's public key; all operations, for all participants, take place in the same group G_q. As such, the selection of key sizes should be driven by the maximum lifetime requirements for both the Issuer's key pair and U-Prove tokens.

When using the elliptic curve construction, the level of security is believed to be the same as in ECDSA (also see [FIPS 186-3]). For general guidelines on picking key sizes to ensure the infeasibility of computing discrete logarithms in G_q, see Section 5.6 of [SP 800-57].

3.2 Hash algorithm selection

The hash algorithm specified in the Issuer parameters MUST be a cryptographically secure hash algorithm, meaning that it MUST be collision-intractable and behave as much as possible as a "random oracle." Furthermore, the digest size of the hash algorithm SHOULD be close or equal to the size of q. A good choice would be the hash function from the SHA family (see [FIPS 180-4]) matching the size of q. In light of state-of-the-art attacks on hash functions, a 256-bit q is RECOMMENDED.

3.3 Random number generation

The strength of the security and privacy properties of U-Prove tokens for each protocol participant critically depend on both the quality and secrecy of the random numbers used by that participant in the protocols. For general guidance on how to generate strong random numbers, see [SP800-90].

3.4 Token lifetime

Issuers can limit the lifetime of U-Prove tokens by encoding a validity period into them, for example in the token information field. Alternatively, a Verifier MAY request that a U-Prove token be obtained in real-time from the Issuer to guarantee the freshness of the attribute data. For security reasons, the Verifier SHOULD request that the presentation message m be encoded into the token;[17] for privacy reasons, the message SHOULD be encoded in the Prover information field.[18]

3.5 Transferability disincentive

An Issuer cannot cryptographically prevent a Prover from transferring a U-Prove token to another party, but it can provide a disincentive for doing so. Since all attribute values A_i are needed to generate a presentation proof, an Issuer could encode confidential information into one special attribute that is never disclosed. The attribute must be encoded directly, i.e., the corresponding Issuer parameter's encoding type e_i must be set to 0x00.[19] To transfer a U-Prove token, the Prover would need to share this confidential information along with the private key.

[17] If the message m is unpredictable, then the Prover cannot obtain U-Prove tokens in advance.
[18] This is to prevent the Issuer and Verifier from linking the issuance and presentation protocols based on the message m.
[19] Otherwise, the Prover could lend the token by sharing x_i without disclosing the confidential hash input A_i.

Alternatively, Device-protected tokens offer strong protection against unauthorized token transfer.

3.6 Trust management

The Issuer parameters are conceptually equivalent to the certificate of a Certificate Authority in a PKI. This specification does not define how trust is established among protocol participants. Conventional techniques can be used to securely share the Issuer and Device parameters with Provers and Verifiers; e.g., they could be signed with a key corresponding to a trusted X.509 CA certificate. Issuer and Device parameters could be obtained before the issuance or presentation protocol, downloaded from a trusted directory, or preinstalled by participants.

3.7 Device-protected tokens

When using Device-protected tokens, it is RECOMMENDED to use a recommended group and the associated Device generator g_d. Moreover, Issuers SHOULD only accept authenticated Device public keys h_d to make sure they are well-formed and belong to the user; otherwise users might try to modify token attributes values by providing a specially-crafted Device public key.

For most operations, the Prover does not need to entrust their privacy or security to the Device, since the protocol is robust to Devices that deviate from the protocol (either maliciously or accidentally). However, when a scope-exclusive device pseudonym is used, the Prover sends the value $P_s = g_s^{x_d}$ to the Issuer without modification. In this case, collusion between the Device and Issuer, sharing knowledge of x_d can identify the user. In applications where such collusion is possible, device pseudonyms SHOULD NOT be used.

3.8 Parallel issuance

The authors of [BLLOR] have recently showed that many parallel issuance sessions for the same attribute values could allow an attacker to create an extra valid token. This is not an issue for use cases where an unlimited number of tokens can be issued to the Prover (like in most identity scenarios); this could however be problematic when tokens have a value (e.g., e-coins, tickets, etc.), and additionally do not have an attribute that is unique per token. If this is the case, implementations SHOULD issue tokens in a sequential manner.

Acknowledgments

The authors would like to thank Stefan Brands, Lan Nguyen and Melissa Chase for their input on the specification.

References

[ANSI X9.62] American National Standard for Financial Services. *X9.62 - 1998. Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*, January 7, 1999.

[BLLOR] Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, Mariana Raykova. On the (in)security of ROS. Journal of Cryptology, September 2022. https://eprint.iacr.org/2020/945.pdf.

[Brands] Stefan Brands. *Rethinking Public Key Infrastructures and Digital Certificates.* The MIT Press, August 2000. http://www.credentica.com/the_mit_pressbook.html.

[FIPS180-4] NIST. *FIPS 180-4 Secure Hash Standard*, March 2012. http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf.

[RFC2119] Scott Bradner. *RFC 2119: Key words for use in RFCs to Indicate Requirement Levels*, 1997. ftp://ftp.rfc-editor.org/in-notes/rfc2119.txt.

[SEC1] Certicom Research, *Standard for Efficient Cryptography 1: Elliptic Curve Cryptography*, May 2009. http://www.secg.org/download/aid-780/sec1-v2.pdf.

[SP800-57] NIST. *SP 800-57 Recommendation for Key Management - Part 1: General (Revision 3)*, July 2012. http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf.

[SP800-90] NIST. *SP 800-90 Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised)*, January 2012, http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf.

[UPCTV] *U-Prove Cryptographic Test Vectors V1.1 (Revision 3)*. Microsoft, December 2013. http://www.microsoft.com/u-prove.

[UPRPP] Christian Paquin. *U-Prove Recommended Parameters Profile V1.1 (Revision 2)*. Microsoft, April 2013. http://www.microsoft.com/u-prove.

[UPTO] Christian Paquin. *U-Prove Technology Overview V1.1 (Revision 2)*. Microsoft, April 2013. http://www.microsoft.com/u-prove.