U-Prove Equality Proof Extension

Draft Revision 1

Microsoft Research

Author: Mira Belenkiy

June 2014

Summary

This document extends the U-Prove Cryptographic Specification [UPCS] by specifying equality of discrete logarithm representation proofs. This allows proving equality between U-Prove attribute values.

Contents

List of Figures

Change history

Version	Description
Revision 1	Initial draft

1 Introduction

This document extends the U-Prove Cryptographic Specification [UPCS] by specifying equality of discrete logarithm representation proofs. This allows proving equality between U-Prove attribute values.

The Prover and Verifier have as common input a list of values $A_0, A_1, \dots, A_{n-1} \in G_q$ and a list of generators $g_{i,0}, g_{i,1}, \dots, g_{i,n_i-1} \in G_q$ corresponding to each A_i. The Prover will create a special-honest verifier zero-knowledge proof of knowledge of the discrete logarithm representation of all the A_i in terms of the generators:

$$\pi = PK\left\{\{\alpha_{i,j}\}_{i\in[0,n-1],j\in[0,n_i-1]} \middle| \forall i \in [0, n-1]: A_i = \prod_{j=0}^{n_i-1} g_{i,j}^{\alpha_{i,j}}\right\}$$

We will call each statement of the form $A_i = \prod_{j=0}^{n_i-1} g_{i,j}^{\alpha_{i,j}}$ a DL equation. (Pedersen Commitments $A = g^{\alpha}h^{\beta}$ are a special case of DL equations.) The Prover has as input a list of open DL equations; i.e. the Prover knows the value of all the $\{\alpha_{i,j}\}_{i\in[0,n-1],j\in[0,n_i-1]}$. The Verifier has as input a list of closed DL equations; i.e. the Verifier only knows the constants $A_0, A_2, \dots, A_{n-1} \in G_q$ and generators $g_{i,0}, g_{i,1}, \dots, g_{i,n-1} \in G_q$.

The Prover and Verifier also have as common input an equality map $\mathcal{M}$ that describes which of the exponents $\alpha_{i,j}$ are equal. The equality map consists of a sorted dictionary, keyed by named variables. Each named variable is associated with a list of exponent indices. For example, the key $(beta, 1)$ could be associated with the list $(0,3), (2,4), (5,1)$. This means that $\propto_{0,3}=\propto_{2,4}=\propto_{5,1}$. The name $(beta, 1)$ is an arbitrary label for the variable associated with these exponents.

Suppose the Prover wants to show that the values A_0 and A_1 have the same discrete logarithms

$$\pi = PK\{\propto_{0,0}, \propto_{1,0} | A_0 = g_{0,0}^{\propto_{0,0}} \cap A_1 = g_{1,0}^{\propto_{1,0}}\}$$

The Prover needs to append a map to the proof indicating that $\propto_{0,0}=\propto_{1,0}$. The Prover assigns the name $name = (gamma, 0)$ to the variable and places the indices of these two exponents in a list: $list = ((0,1), (1,1))$. The resulting equality map would contain a single entry: $\mathcal{M} = ((name, list))$.

The U-Prove Cryptographic Specification [UPCS] allows the Prover, during the token presentation protocol, to create a Pedersen Commitment and show that the committed value is the equal to a particular token attribute. The Prover MAY use this Pedersen Commitment as a DL equation for the equality proof. The Issuance and Token Presentation protocols are unaffected by this extension. The Prover may choose to create an equality proof after these two protocols complete.

1.1 Notation

In addition to the notation defined in [UPCS], the following notation is used throughout the document.

A_i	Value of the i[th] DL equation
$\propto_{i,j}$	Exponent for the i[th] DL equation for j[th] generator $g_{i,j}$.
$x_{i,j}$	Value of $\propto_{i,j}$ known only to the Prover.
$g_{i,j}$	The j[th] generator for the i[th] DL equation.
$\mathcal{M}$	Equality map.
M	Number of entries in equality map.
$name_m$	Name of variable m in the equality map.

$list_m$	List of exponents associated with $name_m$ in the equality map.
n_i	Number of generators/exponents in the i[th] DL equation.
n	Total number of DL equations.
b_i	Part of equality proof: "commitment".
c	Part of equality proof: "challenge".
r_i	Part of set membership proof: "response".

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 Feature overview

The Prover and Verifier have as common input a list of closed DL equations of the form:

$$\left\{\{\alpha_{i,j}\}_{i\in[0,n-1],j\in[0,n_i-1]}\middle|\forall i \in [0,n-1]: A_i = \prod_{j=0}^{n_i-1} g_{i,j}^{\alpha_{i,j}}\right\}$$

They also have as common input an equality map $\mathcal{M} = \left((name_0, list_0), \dots, (name_{M-1}, list_{M-1})\right)$. Each $name_m$ is an arbitrary label for a variable; the equality map is keyed and sorted by the $name_m$. Each $list_m$ contains a sequence of tuples (i,j) indicating exponents $\alpha_{i,j}$. Together, the closed DL equations and equality map constitute the *proof statement*.

The Prover knows a *witness* to the proof statement: a list of values $x_{i,j} = \propto_{i,j}$ that would satisfy the DL equations and equality map.

We quickly overview the equality proof protocol. Suppose the Prover wants to prove the simple statement

$$\pi = PK\{\propto_{0,0}, \propto_{1,0} | A_0 = g_{0,0}^{\propto_{0,0}} \cap A_1 = g_{1,0}^{\propto_{1,0}}\}$$

The Prover knows a pair of witnesses $(x_{0,0}, x_{1,0}) = (\alpha_{0,0}, \propto_{1,0})$. The Prover would perform the following steps:

1. Choose random $w_{0,0}, w_{1,0} \leftarrow \mathbb{Z}_q$ and compute commitments $b_0 := g_{0,0}{}^{w_{0,0}}$, $b_1 := g_{1,0}{}^{w_{1,0}}$.
2. Compute the challenge $c = H(A, g, b_0, b_1)$.
3. Compute the responses $r_{0,0} = w_{0,0} - cx_{0,0} \bmod q$ and $r_{1,0} = w_{1,0} - cx_{1,0} \bmod q$.

The proof consists of $(b_0, b_1, r_{0,0}, r_{1,0})$. The Verifier would check that:

$$b_0 = A_0^c g_{0,0}^{r_{0,0}} \cap b_1 = A_1^c g_{1,0}^{r_{1,0}}$$

Multiple Exponents. If the Prover wishes to prove knowledge of a representation of A_i using n generators and exponents, the Prover would choose $w_{i,0}, \dots, w_{i,n-1} \leftarrow \mathbb{Z}_q$ and compute the commitment $b_i := \prod_{j\in[0,n-1]} g_{i,j}{}^{w_{i,j}}$. The Prover would compute a separate response $r_{i,j} = w_{i,j} - cx_{i,j} \bmod q$ for each exponent. The verification equations would be modified in the obvious way to include all of the generators and responses.

Equality Map. Suppose the equality map includes an entry $list_5 = \left((i,j),(k,l)\right)$. The Prover would choose a random value $w_5 \leftarrow \mathbb{Z}_q$ and use it to compute the commitments b_i and b_k by replacing $w_{i,j}$ and $w_{k,l}$ with w_5 in the product. Similarly, instead of computing two separate responses $r_{i,j}$ and $r_{k,l}$, the Prover would compute a single response $r_5 = w_5 - cx_{i,j} \bmod q$. The Verifier would use the equality map to determine where to use the response r_5 in the verification equation.

2 Protocol specification

As the equality proof can be performed independently of the U-Prove token presentation protocols, the common parameters consist simply of the group G_q and a cryptographic function $\mathcal{H}$.

2.1 Equality Map

The equality map tells the Prover and Verifier which of the exponents in the proof are equal. The equality map has the following grammar:

Equality Map Grammar

$$\mathcal{M} \coloneqq \varepsilon|(Variable)$$
$$Variable \coloneqq (variable\_name, Index\_List)|(variable\_name, Index\_List), Variable$$
$$Index\_List \coloneqq index, index|index, Index\_List$$
$$variable\_name \coloneqq (\text{string,int})$$
$$index \coloneqq (\text{int,int})$$

Figure 1: Equality map grammar

The equality map is a list of zero, one, or more Variables. Each Variable consists of a variable_name and an Index_List. The variable_name is a pair consisting of a string and an integer – e.g. (beta, 4). The Index_List is a list of Index elements indicating which exponents $\alpha_{i,j}$ are equal to each other. Each Index is a pair of integers – e.g. (6,2). The first integer in an Index is the index of a DL Equation while the second integer is the index of the exponent.

The equality map MUST be sorted lexicographically by variable_name. To compare two variable_names, first compare the string element and then, if they are equal, compare the integer element. The same variable_name MUST NOT appear more than once in the equality map.

An Index_List associated with a variable_name must contain at least two elements.

The equality map MUST support a method GetIndex that takes as input an Index (i, j) and outputs the index of the variable_name, with -1 on failure. We will use the short-hand $(i, j) \in \mathcal{M}$ to indicate $\text{GetIndex}(\mathcal{M}, i, j)$ returns an answer greater than -1 and $(i, j) \notin \mathcal{M}$ to indicate $\text{GetIndex}(\mathcal{M}, i, j)$ returns -1. Figure 2 shows a sample implementation of GetIndex.

GetIndex( )

Input

Equality Map: $\mathcal{M}$

Index: i, j

Computation

For all $m \in [0, M-1]$

    If $(i, j) \in list_m$ **return** m

end

return -1

Output

m

Figure 2: GetIndex

2.2 Presentation

The Presentation protocol is shown in Figure 3. Note that for efficiency, it is important to compute each b_i in a single multi-exponentiation rather than a sequence of n_i individual exponentiations.

EqualityProve( )

Input

Parameters: $desc(G_q)$, $\mathrm{UID}_{\mathcal{H}}$

List of DL Equations: $\{A_i, g_{i,0}, g_{i,1}, \dots, g_{i,n_i-1}\}_{i\in[0,n-1]}$

Equality Map: $\mathcal{M}$

Witness: $\{x_{i,j}\}_{i\in[0,n-1],j\in[0,n_i-1]}$

Computation

Choose random $w_0, w_1, \dots, w_{M-1} \leftarrow \mathbb{Z}_q^*$
For all $i \in [0, n-1]$
  For all $j \in [0, n_i - 1]$
    $m := \mathrm{GetIndex}(\mathcal{M}, i, j)$
    If $m > -1$ then
      $w_{i,j} := w_m$
    else
      Choose random $w_{i,j} \leftarrow \mathbb{Z}_q^*$
    end
  end
  $b_i := \prod_{j\in[0,n_i-1]} g_{i,j}^{w_{i,j}}$
end
$c := \mathcal{H}\left(desc(G_q), \{A_i, g_{i,0}, g_{i,1}, \dots, g_{i,n_i-1}\}_{i\in[0,n-1]}, \{b_i\}_{i\in[0,n-1]}\right)$
For all $i \in [0, n-1]$
  For all $j \in [0, n_i - 1]$
    $m := \mathrm{GetIndex}(\mathcal{M}, i, j)$
    If $m > -1$ then
      $r_m := w_m - cx_{i,j} \bmod q$
    else
      $r_{i,j} := w_{i,j} - cx_{i,j} \bmod q$
    end
  end
end

Output

Return $\left(b_0, \dots, b_{n-1}, r_0, \dots, r_{M-1}, \{r_{i,j}\}_{(i,j)\notin\mathcal{M}}\right)$

Figure 3: EqualityProve

2.3 Verification

The Verification protocol is shown in Figure 4. Note that for efficiency, it is important to compute each d_i in a single multi-exponentiation rather than a sequence of $n_i + 1$ individual exponentiations.

EqualityVerify( )

Input

Parameters: $desc(G_q)$, $\mathrm{UID}_{\mathcal{H}}$

List of DL Equations: $\{A_i, g_{i,0}, g_{i,1}, \dots, g_{i,n_i-1}\}_{i\in[0,n-1]}$

Equality Map: $\mathcal{M}$

Proof: $\left(b_0, \dots, b_{n-1}, r_0, \dots, r_{M-1}, \{r_{i,j}\}_{(i,j)\notin\mathcal{M}}\right)$

Computation

$c \coloneqq \mathcal{H}\left(desc(G_q), \{A_i, g_{i,0}, g_{i,1}, \dots, g_{i,n_i-1}\}_{i\in[0,n-1]}, \{b_i\}_{i\in[0,n-1]}\right)$

$pass \coloneqq true$

For all $i \in [0, n-1]$

  For all $j \in [0, n_i - 1]$

    $m \coloneqq \mathrm{GetIndex}(\mathcal{M}, i, j)$

    If $m > -1$ then

      $v_{i,j} \coloneqq r_m$

    else

      $v_{i,j} \coloneqq r_{i,j}$

    end

  end

  $d_i \coloneqq A_i^c \cdot \prod_{j\in[0,n_i-1]} g_{i,j}^{v_{i,j}}$

  If $b_i \neq d_i$ then $pass \coloneqq false$

end

Output

Return *pass*

Figure 4: EqualityVerify

3 Security considerations

The equality proof protocol is a standard Sigma protocol transformed using the Fiat-Shamir heuristic into a non-interactive proof. The following restrictions apply:

1. The Prover and the Verifier MUST NOT know the relative discrete logarithm of any of the generators $g_{i,0}, g_{i,2}, \dots, g_{i,n-0} \in G_q$ that are part of the same DL equation. The Prover and Verifier MAY know the relative discrete logarithm of $g_{i,j}$ and $g_{k,l}$ only if $i \neq k$. This is not an issue if all generators are chosen from the list of U-Prove recommended parameters. However, the generators MAY be chosen from some other set (i.e. as part of some greater protocol) or reused for different DL equations.
2. The equality map has following constraints:
   a. A tuple (i,j) MUST NOT appear more than once in $\mathcal{M}$. Specifically, (i,j) may not be listed as part of the same $list_m$ more than once, and (i,j) may not belong to more than one list.

b. A tuple $(i,*)$ MUST NOT appear more than once in the same $list_m$. Specifically, if $(i,a) \in list_m$ then $(i,b) \notin list_m$.

References

[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.

[UPCS] Christian Paquin, Greg Zaverucha. *U-Prove Cryptographic Specification V1.1 (Revision 3)*. Microsoft, December 2013. http://www.microsoft.com/u-prove.