Tracing Attacks on U-Prove with Revocation Mechanism *

Lucjan Hanzlik, Przemysław Kubiak, Mirosław Kutyłowski
Faculty of Fundamental Problems of Technology, Wrocław University of Technology
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland
firstname.secondname@pwr.edu.pl

ABSTRACT

Anonymous credential systems have to provide strong privacy protection: a user may prove his (chosen) attributes without leaking neither his identity nor other attributes. In this paper we consider U-Prove – one of the major commercial anonymous credential systems.

We show that the revocation mechanism designed for U-Prove enables a system provider to efficiently trace the users' activities. Namely, the Revocation Authority run the system provider may execute the U-Prove protocol in a malicious way so that: (a) the deviations from the protocol remain undetected, (b) the Revocation Authority becomes aware of each single authentication of a user in the whole system and can link them (regardless which attributes are disclosed by the user against the verifiers), (c) it can link presentation tokens with the corresponding token issuing procedure (under some conditions).

Thereby, the system described in the technical drafts of U-Prove does not guarantee privacy protection unless the system provider can be trusted unconditionally. In fact, a malicious provider may convert the Revocation Authority into a "Big Brother" installation.

Keywords

U-Prove, anonymous credential, revocation, malicious provider

1. INTRODUCTION

Emergence of a global information IT ecosystem creates today severe privacy protection challenges. The ecosystem is composed of a large number of systems run by different players that are independent, have different goals, operate within different legal frameworks and sometimes are even prohibited by law to orchestrate their activities. Moreover, from the technical point of view, the scale and limited control over the information flow make it almost infeasible to protect a user against misuse of personal data.

*This is a full version of the paper accepted for presentation at ACM AsiaCCS'15.

It seems that the most effective approach to reduce the scale of privacy protection problems is to reduce the amount of personal data that are revealed during interactions with IT systems. The basic paradigm is that only the data necessary to perform a given action are revealed during the interaction. The traditional approach is completely different: at the very beginning we have to show details about our identity. Changing this situation is a challenge from cryptographic point of view: we have to develop a simple and user friendly scheme that guarantees reliable authentication of user's access rights without disclosing his identity. On the other hand, such solution would fulfill the dream of *privacy-by-design* – which is a political target of European Union in the area of personal data protection.

Many cryptographic schemes have been developed for this purpose, usually called the same name *anonymous credentials* despite substantial functional differences. Most of them have not reached practical maturity stage and have not been implemented. The remarkable exceptions are *Idemix* of IBM Zürich [1], *U-Prove* of Microsoft Research [4] and a simplified but efficient scheme implemented on German personal identity cards under the name *Restricted Identification*.

Anonymous credentials. Anonymous credential system is a cryptographic framework in which a person receives an authentication token from a trust provider for the system. The token confirms a set of *attributes* of the owner, e.g. legal status, age, the rights to access a certain IT system, privileges, etc., but also the name, personal identity number, and so on. A holder of such a token, say Alice, can use it for authentication. If A is the set of attributes of Alice confirmed by the token, A' is an arbitrary subset of A, then she can execute an authentication protocol with Bob so that:

- she proves Bob that she holds an authentication token with all attributes from A',
- however, Bob cannot conclude anything about the attributes not contained in A', in particular in $A \setminus A'$.

Note that "the attributes not contained in A'" may include identity data such as the first name, the family name, and the personal ID number. The attributes revealed may include a pseudonym of the prover.

Based on the result of authentication protocol (and the value of attributes presented) the verifier Bob can make appropriate decisions.

A good example of an attribute is the legal age enabling to engage into civil contracts.

Thereby, anonymous credentials can be used to reduce the information flow during authentication to the necessary minimum. So, we may hope that anonymous credentials may become a standard authentication method in privacy aware systems fulfilling the "privacy-by-design" requirement.

There are many models of anonymous credentials and subtle differences between them. For instance, it is an open issue whether the verifier should be able to check whether the same (anonymous) person authenticates herself, when the set of attributes is exactly the same. Both possibility of linking (*weak privacy*) and infeasibility of linking (*strong privacy*) correspond to some concrete use cases.

U-Prove. The topic of this paper is an extension of U-Prove anonymous credentials system [4] developed by Microsoft. U-Prove is based on the quite complex ideas of Stefan Brands [5, 6], which evolved into the current anonymous credential system.

On the upside, U-Prove is very flexible and quite advanced regarding privacy protection. On the downside, its design is fairly complex. This is a major disadvantage not because of the implementation pains and hardness to explain the security mechanisms even to specialists, but mainly due to hardware requirements far above the possibilities of cryptographic smart cards. U-Prove is based on hardness of Discrete Logarithm Problem and that by design, U-Prove enables commitments which can be used to extend the specification. An interested reader may refer to the web page [7] for details of the U-Prove scheme and implementation details.

In the meantime, U-Prove became declared as *"fully compatible with ABC4Trust engine"* (ABC4TRUST [12] is a EU funded project that aims to provide a unified framework for privacy protection). A large number of system details has been published [7] enabling external designers to integrate their products with U-Prove. Therefore, U-Prove has to be regarded as a system that has a real chance to play an important role in practice on a global scale, despite of lack of thorough academic evaluation.

Revocation. It may happen that the attributes terminate to be valid for a holder of an authentication token. For instance, the token may be stolen. Then privacy protection offered by the system has a negative side effect – it hides identity of a misbehaving person very effectively. So the advanced technology starts to be dangerous to honest users, as there are more incentives to steal the tokens. These problems has been also recognized by the designers of U-Prove – see a recent report [2]. The core U-Prove system is not equipped with a revocation system – the necessary functionality must be added to it as an extension.

During Financial Cryptography'2013 the first revocation system for U-Prove has been presented [8]. It contained a security flaw [11], but the error seems to be correctable. The major drawback of the revocation system from [8] is heavy use of bilinear pairings. According to [2], *"Although pairings are popular in recent cryptographic research, they are seldom used in practice due to their maturity level and implementation complexity."* Definitely, currently the consumer's hardware devices are not supporting pairings (may be with some small scale exceptions).

Following [8], the Microsoft team proposed a modified revocation system which is not based on pairings and can be implemented in standard groups [9] (see [10] for a slightly updated version).

Our contribution. We investigate the U-Prove revocation extension [9, 10] from the point of view of user's privacy. We show that within the current framework it is impossible to hide the revocation attributes of the users (which are effectively the user's pseudonyms) against the Revocation Authority. Namely, the Revocation Authority may learn about each single authentication performed with the user's pseudonym. For this purpose the Revocation Authority must deviate from the original protocol, but it cannot be detected by the users and the verifiers.

Although each security flaw presented in the paper can be fixed by re-designing the scheme and introducing new (computationally heavy) protocol components, it is not clear whether the extensions do not bring new threats and attack possibilities.

2. U-PROVE REVOCATION SYSTEM

In this section we describe shortly the construction from [10]. We do not present the construction idea – in fact, it has not been explained by the authors of [9, 10] and we could only make some guesses. Also, given the page limit and complexity of the design it is impossible to give a reasonable overview of the background U-Prove system. However, this is not really necessary for describing the attacks against user's privacy. Let us note that the same problems occur for the first draft version [9].

2.1 Parameters

The extension from [10] uses the same Issuer parameters as in the standard U-Prove specification. However, the scheme employs the Revocation Authority (RA) initialized in the following way:

Input:	
group:	a group G_q of a prime order q
generators of G_q from U-Prove [7]:	g, g_1, g_t
Computation:	
choose at random	$\delta \in \mathbb{Z}_q$
compute	$K := g^{\delta}$
Output:	
private key:	δ
public key:	K

Table 1: Setup procedure `RSSetup()` for the Revocation Authority

A user gets a token just as in case of the original U-Prove. Additionally, it contains a revocation attribute x_{id} for the user. Its usage will be explained below.

2.2 Revocation List Management

In order to keep record about the revoked users, [10] applies cryptographic accumulators storing identifiers of the revoked users (similarly to a *blacklist*). The idea is borrowed in a substantial part from [8]. The accumulator values are computed as follows:

Input:	
RA private key:	$\delta \in \mathbb{Z}_q$
revocation parameter:	g_t
List of revoked attribute values:	$R = \{x_1, \ldots, x_m\} \in \mathbb{Z}_q \backslash \{-\delta\}$
Computation:	
Accumulator value:	$V := g_t^{\prod_{i=1}^{m}(\delta + x_i)}$

Table 2: Procedure `ComputeAccumulator()` for creating accumulator V for the list R of revocation attributes

2.3 Computing a Witness

Apart from the accumulator, there are parameters specific to each user. These parameters must be updated every time when some user gets revoked or admitted again to the system. There are two methods described in [10]: `ComputeWitness` and `UpdateWitness`. The major drawback of the second solution is low efficiency and necessity to disclose all revocation attributes of the revoked users. According to [10], *"If the revocation list is secret, or for better efficiency, the witnesses are computed by the Revocation Authority"* – with `ComputeWitness`.

The Revocation Authority runs `ComputeWitness` as follows:

Input:	
RA private key:	$\delta \in \mathbb{Z}_q$
Revocation parameter:	g_t
List of revoked attribute values:	$R = \{x_1, \ldots, x_m\} \in \mathbb{Z}_q \setminus \{-\delta\}$
Target user's revocation attribute:	$x_{id} \notin R$
Current accumulator:	$V \in G_q$
Computation:	
$d := \prod_{x \in R}(x - x_{id}) \bmod q$	
$W := g_t^{(\prod_{x \in R}(\delta + x) - d)/(\delta + x_{id})}$	
$Q := VW^{-x_{id}} g_t^{-d}$	
Output:	
Revocation witness for x_{id}:	(d, W, Q)

Table 3: Procedure `ComputeWitness()`

The following property can be easily checked:

$$W^{\delta} = Q \,. \tag{1}$$

`UpdateWitness` procedure can be executed by the users given all revocation identities that have been included in the accumulator since the last update:

Input:	
Revocation parameter:	$g_t \in G_q$
The revocation attribute of the user:	x_{id}
Revocation attribute to be added or removed from the blacklist R:	x'
Boolean value indicating whether x' has to be added to R:	`add`
Old accumulator:	$V \in G_q$
Old witness of the user holding x_{id}:	(d, W, Q)
Updated accumulator:	$V' \in G_q$
Computation:	
if `add` = `true`	(x' added to R)
$d' := d(x' - x_{id}) \bmod q$	
$W' := VW^{x' - x_{id}}$	
$Q' := V'W'^{-x_{id}} g_t^{-d'}$	
else	(x' removed from R)
$d' := d(x' - x_{id})^{-1} \bmod q$	
$W' := ((V')^{-1}W)^{(x' - x_{id})^{-1}}$	
$Q' := V'W'^{-x_{id}} g_t^{-d'}$	
Output:	
Updated witness for x_{id}:	(d', W', Q')

Table 4: Procedure `UpdateWitness( )`

2.4 Non-Revocation Proof

A user holding a U-Prove token authenticates himself by executing the regular U-Prove protocol. During this protocol he commits to the attribute x_{id} (the public output of the commitment is $\tilde{c}_{id} = g^{x_{id}} g_1^{\tilde{o}_{id}}$ and the private opening information of this commitment is $(x_{id}, \tilde{o}_{id})$). Additionally, he creates a non-revocation proof using the function `GenerateNonRevocationProof`:

Input:	
Revocation parameters:	G_q, hash function $\mathcal{H}$, g, g_1, g_t
Commitment to x_{id}:	$\tilde{c}_{id}$, where $\tilde{c}_{id} = g^{x_{id}} g_1^{\tilde{o}_{id}}$
Opening information:	x_{id} , $\tilde{o}_{id}$
RA public key:	K
Revocation witness:	(d, W, Q)
Computation:	
generate $t_1, t_2, k_1, \ldots, k_6$ at random from $\mathbb{Z}_q$	
$X := Wg^{t_1}$	
$Y := QK^{t_1}$	
$C_d := g_t^d g_1^{t_2}$	
$w := d^{-1} \bmod q$	
$z := t_1 \tilde{o}_{id} - t_2 \bmod q$	
$z' := -t_2 w \bmod q$	
$T_1 := X^{k_1} (\tilde{c}_{id} K)^{-k_2} g_1^{k_3}$	
$T_2 := g^{k_1} g_1^{k_4}$	
$T_3 := C_d^{k_5} g_1^{k_6}$	
$c' := \mathcal{H}(g, g_1, g_t, K, \tilde{c}_{id}, X, Y, C_d, T_1, T_2, T_3)$	
$s_1 := -c' x_{id} + k_1 \bmod q$	
$s_2 := -c' t_1 + k_2 \bmod q$	
$s_3 := -c' z + k_3 \bmod q$	
$s_4 := -c' \tilde{o}_{id} + k_4 \bmod q$	
$s_5 := -c' w + k_5 \bmod q$	
$s_6 := -c' z' + k_6 \bmod q$	
delete $t_1, t_2, k_1, ..., k_6, w, z, z', T_1, T_2, T_3$	
Output:	
non-revocation proof for x_{id}:	$c', s_1, \ldots, s_6, X, Y, C_d$

Table 5: Procedure `GenerateNonRevocationProof()`

The non-revocation proof is given to the Verifier. [9] states that *"The Verifier sends the non-revocation proof to the Revocation Authority"*, and that *"The Revocation Authority is a new party that manages the revocation list and validates the users' non-revocation proofs"*. In particular the test $Y \stackrel{?}{=} X^{\delta}$ is delegated to the Revocation Authority. In fact, this is one of the main differences between the design from [8] and [9]. According to [8] the equality $Y = X^{\delta}$ can be checked locally as pairing-friendly groups are used. Similarly, [10] states that *The Verifier sends the non-revocation proof to the Revocation Authority that verifies that the undisclosed UID does not appear on the current revocation list.*

Input:	
Revocation parameters:	$G_q, \mathcal{H}, g, g_1, g_t$
Commitment to x_{id}:	$\tilde{c}_{id}$
Non-revocation proof:	$c', s_1, \ldots, s_6, X, Y, C_d$
Revocation Authority public key:	K
Revocation Authority private key:	δ
Revocation accumulator:	V
Computation:	
$T_1 := (VY^{-1}(C_d)^{-1})^{c'} X^{s_1} (\tilde{c}_{id} K)^{-s_2} g_1^{s_3}$	
$T_2 := \tilde{c}_{id}^{c'} g^{s_1} g_1^{s_4}$	
$T_3 := g_t^{c'} (C_d)^{s_5} g_1^{s_6}$	
verify that $c' = \mathcal{H}(g, g_1, g_t, K, \tilde{c}_{id}, X, Y, C_d, T_1, T_2, T_3)$	
verify that $Y = X^{\delta}$	

Table 6: Procedure `VerifyNonRevocationProof()`

3. BREAKING SECRECY OF REVOCATION LIST

According to Sect. 2.3 of [9, 10], the revocation attributes of the revoked users can be kept secret, if the revocation witnesses are computed and delivered to the users by the Revocation Authority. For some reason the authors of the extension believe that this might be a useful functionality. One of them could be that disclosing the number of revoked users might be a valuable business information. Nevertheless, in this section we show the following result which shows that the attempts to hide the revocation list r are futile:

THEOREM 1. *Assume that the protocol from [9, 10] is implemented as it is stated in the specification. Then even if the Revocation Authority delivers the revocation witnesses to the users, an adversary can retrieve all revocation attributes of the revoked users.*

The point is that the documents [9, 10] do not provide any mechanism designed for preserving secrecy of the revocation list R. We show that the revocation attributes belonging to R cannot be kept secret in a straightforward manner (i.e., by not publishing them), and some additional mechanism is necessary if their secrecy is an objective. Let us note that this section as a warm-up before the main results of Sect. 4, where fundamental privacy threats are shown. However, as it will become clear at the end of Sect. 4, hiding the revocation list is somehow related to hiding malicious setup of the Revocation Authority.

3.1 Reconstruction of the Revocation List

We concern a late launch attack, where the adversary decides to recover the identities of all users revoked so far, but has no data concerning all previous updates performed by the Revocation Authority. (This situation occurs when the adversary joins the system relatively late.) We show that nevertheless an adversary can recover identities of all users revoked assuming that sufficiently many of the users collaborate with the adversary.

Let us assume that at the moment considered the total number of revoked users is k and the set $\mathcal{U}$ of users colluding with the adversary has cardinality at least $k+1$. Let $x_1, \ldots, x_{k+1}$ be the revocation attributes of these users. Then we claim that the adversary can reconstruct the following polynomial:

$$f(X) = (X + x_{i_1}) \cdot (X + x_{i_2}) \cdot \cdots \cdot (X + x_{i_k}), \qquad (2)$$

where $x_{i_1}, ..., x_{i_k}$ are the revocation attributes of all revoked users. Since factorization in $\mathbb{Z}_q[X]$ is easy, once f is found, then the values of x_{i_j} can be easily reconstructed.

First observe that according to `ComputeWitness` procedure, the user holding x_i obtains a revocation witness that contains $d_i = f(-x_i)$. Hence, in order to reconstruct the polynomial f it suffices to run the Lagrangian interpolation algorithm on the pairs $(-x_1, d_1), \ldots, (-x_{k+1}, d_{k+1})$. [1]

3.2 Attacking update's data

The Revocation Authority may defend itself against the above mentioned attack by keeping a large number of dummy users in the set R of revoked users. In this case $k = \deg f$ becomes large and the adversary is unable to find $k+1$ users to immediately gather $k+1$ pairs (x_i, d_i) satisfying $f(-x_i) = d_i$. In this case the adversary may execute the following attack aiming to recover the revocation attributes of the users that are either revoked or admitted again during the current update.

Assume that a user holding x_i and a witness $(d_i, -, -)$ corresponding to the set of revoked users R obtains a new witness d_i'. Then:

$$\frac{d_i'}{d_i} = \frac{P_A(x_i)}{Q_B(x_i)}, \quad \text{where}$$
$$P_A(X) = \textstyle\prod_{x' \in A}(x' - X), \quad Q_B(X) = \prod_{x' \in B}(x' - X)$$

and A denotes the set of revocation attributes that have been added to R since the last update of users' witnesses, whereas B denotes the set of revocation attributes that have been removed from R since this moment. Since A and B are disjoint, $\gcd(P_A(X), Q_B(X)) = 1$. That is, the user gets a value $f_i = d_i'/d_i$ of some rational function at point x_i:

$$\frac{P_A(x_i)}{Q_B(x_i)} = f_i, \qquad (3)$$

where $\deg P_A(X) = |A| \geq 0$, $\deg Q_B(X) = |B| \geq 0$.

Let $\mathcal{U}$ be the set of users controlled by the adversary aiming to get the revocation attributes of the revoked users. When the update is done by the Revocation Authority, each user gets a new revocation witness. So the adversary can derive the pair (x_i, f_i) for each $x_i \in \mathcal{U}$. The adversary shall use these pairs to interpolate the rational function

$$\frac{P_A(X)}{Q_B(X)} = \frac{p_0 + p_1 X + \cdots + p_m X^m}{q_0 + q_1 X + \cdots + q_n X^n}, \qquad (4)$$

where $m = |A|$ and $n = |B|$. The adversary aims to calculate both $P_A(X)$ and $Q_B(X)$. Once it is done, it is easy to factorize $P_A(X)$ and $Q_B(X)$ in the polynomial ring and learn the revocation attributes of the users revoked and admitted back to the system.

Let $k = |\mathcal{U}|$. Assume that m and n have been guessed correctly. By (3) and (4), we can build the following system of linear equations

$$\begin{cases} p_0 + p_1 x_1 + \ldots + p_m x_1^m - f_1 q_0 - f_1 q_1 x_1 - \ldots - f_1 q_n x_1^n = 0 \\ p_0 + p_1 x_2 + \ldots + p_m x_2^m - f_2 q_0 - f_2 q_1 x_2 - \ldots - f_2 q_n x_2^n = 0 \\ \qquad \ldots \\ p_0 + p_1 x_k + \ldots + p_m x_k^m - f_k q_0 - f_k q_1 x_k - \ldots - f_k q_n x_k^n = 0 \end{cases} \qquad (5)$$

with unknowns $p_0, p_1, \ldots, p_m, q_0, q_1, \ldots, q_n$. The system (5) can be rewritten as

$$M_{k,m+n+2} \cdot U_{m+n+2} = \theta_k,$$

where θ_k is the vector of k zeroes,

$$U_{m+n+2} = [p_0, p_1, p_2, \ldots, p_m, q_0, q_1, q_2, \ldots, q_n]^T$$

[1] The same method works for the scheme from [9], as the values of the components d_i are exactly the same despite different descriptions of `ComputeWitness`.

is the vector of $m+n+2$ unknowns, and $M_{k,m+n+2}$ is a $k\times(m+n+2)$ matrix of known coefficients from $\mathbb{Z}_q$:

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^m & -f_1 & -f_1x_1 & -f_1x_1^2 & \dots & -f_1x_1^n \\ 1 & x_2 & x_2^2 & \dots & x_2^m & -f_2 & -f_2x_2 & -f_2x_2^2 & \dots & -f_2x_2^n \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & x_k & x_k^2 & \dots & x_k^m & -f_k & -f_kx_k & -f_kx_k^2 & \dots & -f_kx_k^n \end{bmatrix}$$

Since the system (5) is homogeneous we treat the set of solutions of the form $\alpha\cdot U_{m+n+2}$, where $\alpha\in\mathbb{Z}_q\setminus\{0\}$, as a single solution U_{m+n+2}.

Note that $M_{k,m+n+2}=[V_{k,m+1}\quad F_{k,k}\cdot V_{k,n+1}]$,where $F_{k,k}=\mathrm{diag}(-f_1,-f_2,...,-f_k)$ is the diagonal matrix of size $k\times k$, and

$$V_{k,\ell+1}=\begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^\ell \\ 1 & x_2 & x_2^2 & \dots & x_2^\ell \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_k & x_k^2 & \dots & x_k^\ell \end{bmatrix}$$

is the Vandermonde matrix of size $k\times(\ell+1)$.

Obviously, $\mathrm{rank}(M_{k,m+n+2})\le m+n+2$. Hence, even if $k>m+n+2$, it is sufficient to use the first $m+n+2$ rows of the matrix because the next ones give no additional information. If after reducing the matrix to the row echelon form it happens that $\mathrm{rank}(M_{m+n+2,m+n+2})=m+n+2$, then the only solution for U_{m+n+2} is the all zero vector θ_{m+n+2}, which means that our guess for m and n was incorrect. If after reducing the matrix to the row echelon form it turns out that $\mathrm{rank}(M_{m+n+2,m+n+2})<m+n+2$, then we may neglect a row which turns out to be linearly dependent from the other rows and we use the following theorem:

THEOREM 2 (THM 4.3.4 FROM [13]). *Let $M_{m+n+1,m+n+2}=[V_{m+n+1,m+1}\quad F_{m+n+1,m+n+1}\cdot V_{m+n+1,n+1}]$ be a real- or a complex-valued matrix of rank $m+n+1-t$. Then there exists a unique non-zero solution U_{m+n+2} corresponding to polynomials $P_A^*(X)$, $Q_B^*(X)$ with degrees at most $m-t$ and $n-t$. Moreover, all solutions have the form*

$$r(X)=\frac{S(X)P_A^*(X)}{S(X)Q_B^*(X)},\qquad(6)$$

where $S(X)$ is a polynomial of degree at most t.

Although Theorem 2 considers matrices with coefficients from fields of characteristic 0 (which means that by adding 1 we shall never obtain 0), in large prime fields $\mathbb{F}_q$ the event that some matrix coefficient will "accidentally" vanish during calculation of the rank seems to be highly improbable. On the other hand, if q is relatively small, then to minimize the risk of an "accidental underestimation" of the rank one could use matrices with the number of rows greater than $m+n+2$. Consequently, we have the following procedure for finding the polynomials $P_A(X)$, $Q_B(X)$:

1. We make a guess for m',n' being the realistic upper bounds for the polynomials degrees m and n, respectively.

2. We generate the matrix $M_{m'+n'+2,m'+n'+2}$ and reduce it to the row echelon form to calculate its rank (if the rank will be appropriate the row echelon form will facilitate the final computations).

3. If $\mathrm{rank}(M_{m'+n'+2,m'+n'+2})=m'+n'+2$, then we increase both values m',n' and go to the point 2 (alternatively, if for the current values m',n' we have $m'+n'+2=|\mathcal{U}|$, we may increase one of the values m',n' at the cost of the other one).

4. If $\mathrm{rank}(M_{m'+n'+2,m'+n'+2})=m'+n'+1-t$ and $t>0$, then we assign $m':=m'-t$, $n':=n'-t$ and go to the point 2. Since $\gcd(P_A(X),Q_B(X))=1$ we have in equation (6) that $S(X)\equiv 1$.

5. At this point we have $\mathrm{rank}(M_{m'+n'+2,m'+n'+2})=m'+n'+1$. We start to determine the vector $U_{m'+n'+2}$ by assigning 1 to the coordinate which corresponds to the first column of the matrix (counting from the left) that contains a zero on the main diagonal. In this way value one is assigned to the only independent coordinate of vector $U_{m'+n'+2}$. Then on the basis of the row echelon form we determine the remaining $m'+n'+1$ coordinates. From the form of the matrix and from the fact that the system (5) is homogeneous it follows that the only independent coordinate of $U_{m'+n'+2}$ corresponds to the leading coefficient of the polynomial $Q_B(X)$ (recall that $n\le n'$). In this way we obtain the monic polynomials $P_A(X)$, $Q_B(X)$.

Note that the attack described above may sometimes help to reconstruct the polynomial f even if initially its degree is too large. Recall that the revoked users could be admitted back to the system. So it may happen that at some moment the set of users that were revoked at the initial stage of the attack and are still in the revoked set is reduced below the number of users cooperating with the adversary. So, if this happens and if the adversary keeps track on the changes in the set of revoked users, then he can reconstruct the set of the revoked users at each moment.

Experimental results. In order to check relevance of our method (recall that Theorem 2 concerns the fields of characteristic 0), we have implemented the procedure described above using the NTL library [14]. 1000 experiments has been performed for a 256-bit prime q. We have not encountered a matrix $M_{m'+n'+2,m'+n'+2}$ with $\mathrm{rank}(M_{m'+n'+2,m'+n'+2})\le m'+n'+1$, when $m'<m$ or $n'<n$ (note that this would lead to false results concerning the polynomials $P_A(X)$, $Q_B(X)$). Moreover, for any given experiment the point 4 of the procedure has always been executed at most once with $t>0$. That is, when $t>0$ had been encountered, then after the assignment $m':=m'-t$, $n':=n'-t$ at least one of the equalities $m'=m$, $n'=n$ was satisfied.

4. TRACING USERS' ACTIVITIES

In this section we show that after some manipulations on the side of the Revocation Authority the U-Prove system with revocations becomes a perfect "Big Brother" tool for spying the users via non-revocation proofs. **This is the main result of this paper**.

The schemes proposed in [9, 10] are aimed to be suitable for a group G_q being a "standard construction", which means that the group G_q (see Table 1) need not to be pairing-friendly. Consequently, to verify the equality $Y=X^\delta$ in the procedure from Table 6 the private key δ is needed.

Our attacks exploit the coincidence of the following facts:

1. *"The Revocation Authority is a [...] party that manages the revocation list and validates the users' non-revocation proofs".*
2. Within `GenerateNonRevocationProof()` the elements W and Q appear in X and Y respectively in powers known to the Revocation Authority (simply in powers equal to one).

The initial attacks are less demanding computationally. However, the latter attacks seem to be more general: they remain undetectable even if witnesses are updated by the users from the very beginning of the revocation system work, that is, even if the complete list of the revoked revocation attributes is public. The attacks are described below for [10], however they work for [9] after only minor changes.

4.1 Updates Done by Revocation Authority

In this section we assume that the Revocation Authority periodically provides updated revocation witnesses to the users computed with `ComputeWitness`. According to Sect. 2.3 of [10], *"If the revocation list is secret, or for better efficiency, the witnesses are computed by the Revocation Authority ..."*. Hence, the situation described is one of the scenarios proposed.

The idea is to provide corrupted revocation witnesses. They lead to failure of the equality $X^\delta = Y$, but at the same time the Revocation Authority will be able to verify that X and Y has been created according to the `GenerateaNonRevocationProof` procedure. Moreover, the Revocation Authority will be able to identify the revocation attribute of the user that has created X and Y. Therefore, the Revocation Authority will be able to create the correct answer to for the `VerifyNonRevocationProof()` procedure despite the manipulations.

The user is given a witness $(\underline{d}, W, \underline{Q})$ that deviates from the protocol. Namely:

	correct execution	corrupted execution
1.	W	W (according to the specification)
2.	d	$\underline{d} := d + \Delta$ where Δ chosen at random and d is computed according to the specification
3.	Q	$\underline{Q} := V \cdot W^{-x_{id}} \cdot g_t^{-\underline{d}}$
4.	output (d, W, Q)	output witness $(\underline{d}, W, \underline{Q})$

Potentially, manipulation of d might lead to a situation that the verification of the non-revocation proof would fail already in the part that can be executed by the user or by a verifier different from the Revocation Authority. Note that `VerifyNonRevocationProof()` can be executed by anybody except for the test $Y \stackrel{?}{=} X^\delta$. The first test yields the positive result if the arguments for the hash function $\mathcal{H}$ are the same as when c' has been computed. So we require that computing T_1, T_2 and T_3 during `GenerateNonRevocationProof()` and `VerifyNonRevocationProof()` yield the same results. It may be checked that this is the case even if the value d is manipulated as described above (for details see the Appendix, Sect. A).

Note that

$$\frac{Q}{\underline{Q}} = \frac{V \cdot W^{-x_{id}} \cdot g_t^{-d}}{V \cdot W^{-x_{id}} \cdot g_t^{-d-\Delta}} = g_t^{\Delta} \ .$$

Now consider verification of the values X, Y given for inspection to the Revocation Authority and created from a corrupted witness $(\underline{d}, W, \underline{Q})$ given to a user. The Revocation Authority gets the values $X = W \cdot g^{t_1}$ and $\underline{Y} = \underline{Q} \cdot K^{t_1}$. Observe that

$$X^\delta = W^\delta \cdot g^{t_1 \cdot \delta} = W^\delta \cdot K^{t_1} = Q \cdot K^{t_1} = Y$$

where the value Y corresponds to the honest protocol execution. On the other hand, the Revocation Authority gets $\underline{Y} = \underline{Q} \cdot K^{t_1}$. So

$$\frac{X^\delta}{\underline{Y}} = \frac{Q \cdot K^{t_1}}{\underline{Q} \cdot K^{t_1}} = \frac{Q}{\underline{Q}} = g_t^{\Delta} \ .$$

So a malicious Revocation Authority acts, for instance, as follows:

- for a user holding x_{id} create a corrupted witness and store the pair (g_t^{Δ}, x_{id}) in the local database $\mathcal{T}$,
- during verification of the condition $X^\delta = Y$ for X and Y submitted by a verifier check the value $Z := X^\delta / Y$ against the entries in the database. If there is a pair (Z, x_{id}), then the Revocation Authority responds `correct`, otherwise the Revocation Authority answers `false`.
  In the first case the Revocation Authority learns that the user with x_{id} authenticates himself against the verifier that has submitted the query.

As we see, the malicious Revocation Authority can create the full history of all activities of all users. Moreover, the verifier MUST ask the Revocation Authority to verify the equation $X^\delta = Y$. Indeed, if the condition $X^\delta = Y$ would not be verified, then a revoked user could present a valid non-revocation proof. Namely, instead of using the witness (d, W, Q) obtained from the Revocation Authority, the user may create himself a fake witness $(\underline{d}, \underline{W}, \underline{Q})$, where $\underline{d}, \underline{W}$ are chosen at random and $\underline{Q} = V \cdot \underline{W}^{-x_{id}} \cdot g_t^{-\underline{d}}$.

Note that secrecy of the revocation list R is essential for hiding the attack. Otherwise on the basis of formula (2), a user holding x_{id} could reconstruct the polynomial f and check whether $f(-x_{id}) = d$. Accordingly, the Revocation Authority should insist that hiding revocation attributes of the revoked users is necessary for privacy protection and should deploy techniques such using large number of dummy users to defer the attacks presented in Sect. 3.

4.2 Switching to the Public Mode

Now we investigate the scenario that at some moment the system is switched from the mode "Update of Witnesses done by the Revocation Authority" to the mode where the witnesses are recomputed by the users themselves according to the procedure `UpdateWitness`. However, we still assume that the complete set R of revoked users will not be published. We also assume that when a user joins the system the first revocation witness (d, W, Q) is given to the user by the Revocation Authority. This is a pragmatic assumption increasing system's usability and it results from the fact that without knowing the whole set R the users cannot compute initial witnesses by themselves.

The idea of the attack is to provide a corrupted revocation witness when the user joins the system. Just like in Sect. 4.1, the Revocation Authority receives the pairs (X, Y) where $X^\delta \neq Y$. However, the relation between X and Y enables the Revocation Authority to check whether the non-revocation proof has been created correctly and to learn the x_{id} of the user that has created X and Y.

The initial revocation witness (d', W', Q') given to the user holding x_{id} is created as follows:

Input:	
RA private key:	$\delta \in \mathbb{Z}_q$
Revocation parameter:	g_t
List of revoked attribute values:	R
New user's revocation attribute:	$x_{id} \notin R$
Current accumulator:	V
Auxiliary database:	$\mathcal{T}$
Computation:	
1. compute d and W via `ComputeWitness` for R and x_{id}	
2. choose u at random	
3. $\underline{d} := d + u \bmod q$	
4. $\underline{Q} := VW^{-x_{id}} g_t^{-\underline{d}}$	
Output:	
insert (g_t^u, x_{id}) in the database $\mathcal{T}$	
give $(\underline{d}, W, \underline{Q})$ to the user holding x_{id}	

Table 7: Creating a corrupted initial witness

The database $\mathcal{T}$ stores information necessary to link the verification requests with the users. Note that if Q is the non-corrupted value, then $Q = VW^{-x_{id}} g_t^{-d}$. Hence $Q \cdot g_t^{-u} = \underline{Q}$.

For the corrupted witness $(\underline{d}, W, \underline{Q})$ note that:

- the verification test concerning c' succeeds (note that this test can be done by any verifier),
- the Revocation Authority gets the values X, Y for which the regular verification by the Revocation Authority fails in the following way:

  $$X^{\delta} = W^{\delta} g^{t_1 \delta} = QK^{t_1} = \underline{Q} g_t^u K^{t_1} = Y g_t^u \neq Y$$

  However, the Revocation Authority may search for an entry (Z, x_{id}) such that $X^{\delta} \cdot Y^{-1} = Z$. If there is such an entry, then the answer to the verification query is `correct` and as a side effect the Revocation Authority learns x_{id}. Otherwise, the answer is `false`.

Now let us consider the situation after the user holding x_{id} updates the revocation witness because of revocation of x'.

- According to `UpdateWitness`, the new value of the parameter d, called below $\underline{d_1}$, equals

  $$\underline{d_1} = \underline{d}(x' - x_{id}) = (d+u)(x' - x_{id}) = d_1 + u(x' - x_{id}) \,,$$

  where $d_1 = d(x' - x_{id})$ is the correct value that would be obtained for the correct initial witness.
- The value of W will be updated to the correct value W_1, since no manipulated value is applied for the update.
- The new value of Q equals

  $$\begin{aligned} \underline{Q_1} &= VW^{-x_{id}} g_t^{-\underline{d_1}} = VW^{-x_{id}} g_t^{-d_1 - u(x' - x_{id})} \\ &= Q_1 g_t^{-u(x' - x_{id})} \,. \end{aligned}$$

  where Q_1 is the value of Q computed for the correct d_1.

Now, if the verifier presents a pair (X, Y) created by the user holding x_{id}, then

$$X^{\delta} = W_1^{\delta} g^{t_1 \delta} = Q_1 K^{t_1} = \underline{Q_1} g_t^{u(x' - x_{id})} K^{t_1} = Y g_t^{u(x' - x_{id})} \,.$$

So we see that if each entry (Z, x_{id}) of the database $\mathcal{T}$ is converted by the assignment $Z := Z^{x' - x_{id}}$, then the same procedure as before can be used to answer the queries to the revocation authorities stated by the verifiers.

One can easily show that after revoking $x'_1, \ldots, x'_k$ and adding $x''_1, \ldots, x''_m$ the original entry (Z, x_{id}) of database $\mathcal{T}$ becomes converted to (Z', x_{id}), where

$$Z' = Z^{(x'_1 - x_{id}) \cdot \ldots \cdot (x'_k - x_{id}) \cdot (x''_1 - x_{id})^{-1} \cdot \ldots \cdot (x''_m - x_{id})^{-1}}$$

and the same procedure as above for answering the queries to the Revocation Authority can be applied.

4.3 Tracing Independent of Updating Entity

The attack below remains undetected even if the complete list R of revoked revocation attributes is public. This is achieved by not manipulating the parameters used for witnesses computations.

4.3.1 Preliminary Observations

Let us note that the output of the `RASetup()` procedure does not include a proof of equality of the discrete logarithms within the pairs: (g_t, g_t^{δ}), (g, K), where the 1st element in the pair denotes the base element and the 2nd element denotes the exponentiation result. Accordingly, the Revocation Authority may choose the values δ and $\log_g K$ to be different modulo q. Let $K = g^{\tilde{\delta}}$. One can easily see that `VerifyNonRevocationProof()` does not enforce equality of discrete logarithms for (g_t, g_t^{δ}) and (g, K). This fact will be exploited in our attack.

4.3.2 The Attack

We assume that the Revocation Authority knows all the values x_{id} (hence it is able to compute witnesses of all users). The `RSSetup` procedure is executed in a slightly different way. Namely, the Revocation Authority chooses $\tilde{\delta} \neq \delta \bmod q$ at random and sets $K := g^{\tilde{\delta}}$ instead of $K := g^{\delta}$. Note that due to the hardness of the Decisional Diffie-Hellman Problem it is infeasible to detect this manipulation.

During `VerifyNonRevocationProof()` the Revocation Authority, instead of checking equality $Y = X^{\delta}$ executes the following steps:

1. $\widetilde{W} := Y \cdot X^{-\tilde{\delta}}$.
   Note that X and Y were calculated correctly by the prover, then we get $\widetilde{W} = Q \cdot W^{-\tilde{\delta}}$.
2. $\widehat{W} := \widetilde{W}^{\eta}$, where $\eta = (\delta - \tilde{\delta})^{-1} \bmod q$.

Note that both δ and $\tilde{\delta}$ are known to the Revocation Authority. If the witness was correctly computed by the user, then, by (1), the Revocation Authority gets

$$\widehat{W} = \left(W^{\delta - \tilde{\delta}}\right)^{(\delta - \tilde{\delta})^{-1} \bmod q} = W.$$

Note that W depends on x_{id} and can be used to match the non-revocation proof with x_{id}. On the other hand, W changes each time

the list R of revoked users changes. The Revocation Authority may create the database containing the current value of W for each x_{id}. The computation can be easily performed by a SIMD architecture (e.g. a farm of GPU processors).

4.3.3 *Removing Y from the Non-Revocation Proof*

One could attempt to defend against the attack by removing Y from the data passed to the Revocation Authority. Instead, the Authority may calculate a candidate Y_W for Y used by the prover by setting $Y_W = X^{\delta}$ and checking it against c':

$$c' \stackrel{?}{=} \mathcal{H}(g, g_1, g_t, K, \tilde{c}_{id}, X, Y, C_d, T_1, T_2, T_3) \tag{7}$$

However, for each W the Revocation Authority must compute the candidate value

$$Y_W := W^{\delta-\tilde{\delta}} X^{\tilde{\delta}} .$$

Note that for the correct W

$$Y_W = W^{\delta-\tilde{\delta}} \cdot \left(W g^{t_1}\right)^{\tilde{\delta}} = W^{\delta} \cdot \left(g^{\tilde{\delta}}\right)^{t_1} = QK^{t_1} = Y$$

and for such Y_W equality (7) holds. Otherwise, it may hold only via a collision of the hash function $\mathcal{H}$.

4.4 Tracing when g^{δ}=K

To prevent the attack from Subsect. 4.3 one could propose a patch for the system by demanding a proof of equality of discrete logarithms when publishing the public parameters of the Revocation Authority. Below we show that it is not enough, however the attack gets more involved.

During the corrupted `RSSetup` procedure α_1, α_2 are chosen at random so that $\alpha_2 \neq \alpha_1 \bmod q$. Then

$$\delta := (\alpha_1 + \alpha_2) \cdot 2^{-1} \bmod q . \tag{8}$$

If the order q of g is a large prime number, then $2^{-1} \bmod q$ exists. On the other hand, if $\gcd(2, q) > 1$, then it is easy to generalize the reasoning below to the representation $\delta = (\alpha_1 + \cdots + \alpha_\ell) \cdot \ell^{-1} \bmod q$ for some ℓ coprime to q (in this case the definition of Λ_i given below would change to $\Lambda_i := g_t^{(\alpha_1^i + \cdots + \alpha_\ell^i) \cdot \ell^{-1} \bmod q}$). We shall use the following notation:

$$\Lambda_i := g_t^{(\alpha_1^i + \alpha_2^i) \cdot 2^{-1} \bmod q} \quad \text{and} \quad \Delta_i := g_t^{\delta^i} \tag{9}$$

for $i = 0, 1, 2, \ldots$. Note that $\Lambda_0 = \Delta_0 = g_t$ and $\Lambda_1 = \Delta_1$.

Note that in order to get the accumulator V one can first compute the exponent $\prod_{i=1}^{m}(\delta + x_i)$ and then raise g_t to the computed power. An alternative when is to get first the numbers Δ_i, then compute $\prod_{i=1}^{m}(\delta + x_i)$ as a polynomial of δ, i.e. $\prod_{i=1}^{m}(\delta + x_i) = \sum_{i=0}^{m} a_i \cdot \delta^i$, and finally put $V := \prod_{i=0}^{m}(\Delta_i)^{a_i}$. Below we assume that instead of using the vector

$$\mathbf{t} = (\Delta_0, \Delta_1, \Delta_2, \ldots, \Delta_m) \tag{10}$$

the Revocation Authority will use the vector

$$\mathbf{t}' = (\Lambda_0, \Lambda_1, \Lambda_2, \ldots, \Lambda_m) \tag{11}$$

to calculate the value of the accumulator V. Hence the attack below exploits the fact that the protocol from [10] does not include any proof of equality of discrete logarithms between the pairs (Δ_0, Δ_1), (Δ_i, Δ_{i+1}) for $i = 1, 2, \ldots$, where the first element in the pair denotes the base of the logarithm, and the second element denotes the exponentiation result.

4.4.1 *Preliminary Observations.*

Recall that according to [10] users' witnesses are computed by `ComputeWitness()` or `UpdateWitness()`. As we shall see, the first method may be extended to a more direct method of computing W and Q, a method that may be run by (a set of) users. Hence if the second method is available to the users, then both methods should be coherent, i.e. the results obtained by the direct method and by `UpdateWitness()` should be the same.

Updating Witnesses by the Revocation Authority. Suppose that a group of users has reconstructed the polynomial f from (2) as described in Sect.3.1. Then $k-1$ users will be able to find the powers $g_t^{Y_{k-1}}, g_t^{Y_{k-2}}, \ldots, g_t^{Y_1}$ that are used by the Authority to generate the values of the witnesses. Namely, consider the following procedure:

1. A colluding user holding own parameters x_{id}, d_{id} and the global parameter $f(X)$ finds the polynomial

$$g_{id}(X) = (f(X) - d_{id})/(X + x_{id}) = \sum_{j=k-1}^{0} \mu_{id,j} X^j \tag{12}$$

So the user learns all the coefficients $\mu_{id,j}$ in the equation

$$g_t^{\sum_{j=k-1}^{0} \mu_{id,j} Y_j} = W_{id}, \tag{13}$$

where for each $j = k-1, k-2, \ldots, 1$ the unknown Y_j presumably satisfies the condition $Y_j = \delta^j$. We must emphasize that the colluding users assume that in (13) the values W_{id} are indeed calculated according to the protocol. However, $g_t^{Y_j}$ will successfully be determined even if $Y_j \neq \delta^j$ for at least some of the $j = 0, 1, \ldots$.

2. The colluding users gather all $k-1$ equations together and for the system of equations

$$\begin{cases} g_t^{\sum_{j=1}^{k-1} \mu_{id_1,j} Y_j} &= W_{id_1} \cdot g_t^{-\mu_{id_1,0}} \\ g_t^{\sum_{j=1}^{k-1} \mu_{id_2,j} Y_j} &= W_{id_2} \cdot g_t^{-\mu_{id_2,0}} \\ \ldots & \\ g_t^{\sum_{j=1}^{k-1} \mu_{id_{k-1},j} Y_j} &= W_{id_{k-1}} \cdot g_t^{-\mu_{id_{k-1},0}} \end{cases} \tag{14}$$

perform Gaussian elimination in the exponents. That is, instead of scalar multiplications they have exponentiations, instead of subtractions they have divisions in the group generated by g_t. Since x_{id} are assumed to be random and independent, they count for the corresponding matrix (i.e., the matrix $[\mu_{id_i,j}]$, for $i, j \in \{1, \ldots, k-1\}$) to be invertible. [2] Thereby the colluding users should obtain the set of solutions $g_t^{Y_j}$, for $j = k-1, \ldots, 1$.

Of course, if the set of revoked users increases over time to some value k' ($k' > k$), then to find the newly used $g_t^{Y_{k'-1}}, g_t^{Y_{k'-2}}, \ldots, g_t^{Y_k}$ only $k'-k$ colluding users are needed. The same follows for the accumulator V: if $g_t^{Y_{k-1}}, g_t^{Y_{k-2}}, \ldots, g_t^{Y_1}$ are found and $V = Q_{id} \cdot W_{id}^{x_{id}} \cdot g_t^{d_{id}}$ is calculated by each user with the corresponding x_{id} and corresponding witness (d_{id}, W_{id}, Q_{id}), then $g_t^{Y_k}$ is easily calculated by any colluding user from V, $g_t^{Y_{k-1}}, g_t^{Y_{k-2}}, \ldots, g_t^{Y_1}$.

[2] It may happen that the matrix in not invertible, but via a number of experiments we never encountered such a case.

Assume that f is the polynomial (2) from a revocation round, and $\deg f \leq k$. Then knowing f and the powers $g_t^{Y_j}$, for $j = k, \ldots, 1$, each colluding user can calculate the accumulator value and the witness corresponding to that f and x_{id}. Indeed, the user computes

$$d_{id} := f(-x_{id}) \tag{15}$$

and the quotient polynomial (12). Finally, he directly computes:

$$\begin{aligned} W_{id} &:= \prod_{j=k-1}^{0} \left(g_t^{Y_j}\right)^{\mu_{id,j}}, \\ Q_{id} &:= \prod_{j=k-1}^{0} \left(g_t^{Y_{j+1}}\right)^{\mu_{id,j}} \end{aligned} \tag{16}$$

for $\mu_{id,j}$ such that $g_{id}(X) = \sum_{j=k-1}^{0} \mu_{id,j} X^j$. Note that in these computations Q_{id} corresponds to W_{id} shifted by one step to the "higher" powers $g_t^{Y_{j+1}}$ reflecting the (presumed, but in case of the attack not satisfied) relation $Q_{id} = W_{id}^{\delta}$.

Witnesses Updated by the Users. A similar method can be applied in this case. Moreover, if the initial set of revoked users is empty, then all the necessary data for building the direct method are available to each single user (the user may create virtual identities x_{id} to compute their witnesses and thus to gather the data for the system (14)). Once the direct method of computing W_{id} and Q_{id} is obtained, it can be used interchangeably with `UpdateWitness()`.

4.4.2 Coherence of Witness Computation Methods

Some users may sometimes prefer to use the direct method of computing W_{id}, Q_{id} over the `UpdateWitness()` procedure. Consider the exemplary scenarios:

- the degree of polynomial $f(X)$ does not change much but there are many fluctuations on R (some users are revoked but some are admitted back to the system), and a user updating the witness makes the update only occasionally (i.e., only when needed),
- a new client application that include the `UpdateWitness()` procedure undergoes tests and some independent method for computing the witnesses is needed.

Let us take a closer look at the `UpdateWitness()` procedure: The list of the arguments of `UpdateWitness()` contains the value V of the old accumulator and the value V' of the new accumulator. By $\tilde{f}(X)$ let us define the new polynomial being the product of all the binomials $(X + \tilde{x})$ such that $\tilde{x} \in R$. That is, $\tilde{f}(X) = f(X) \cdot (X + x')$, if x' has been added to the set R of revoked values, and $\tilde{f}(X) = f(X)/(X + x')$, if x' has been removed from R. Note that in the latter case $(X + x')$ divides the old polynomial $f(X)$.

If $\deg \tilde{f}$ is greater than the degree of any polynomial f resulting from the set R published so far, then the Revocation Authority is the only party capable to calculate V'. In this way, for consecutive $i = 1, 2, \ldots$, the next value Λ_i from vector (11) is implicitly introduced to the system by the Revocation Authority. (Note that Λ_0 is already a part of the public key of the Revocation Authority). Let

$$\tilde{f}(X) = \sum_{i=k'}^{0} \tilde{a}_i X^i, \qquad f(X) = \sum_{i=k}^{0} a_i X^i,$$

where $k' = \deg \tilde{f}$, $k = \deg f$, $\tilde{a}_i \in \mathbb{Z}_q$ are coefficients of the polynomial $\tilde{f}$, and $a_i \in \mathbb{Z}_q$ are coefficients of the polynomial f.

We have $\tilde{a}_{k'} = 1$ and $a_k = 1$, and $k' = k + 1$ if x' is added to R and $k' = k - 1$ if x' is removed from R. So the users see:

$$V' = \prod_{i=0}^{k'} \Lambda_i^{\tilde{a}_i}, \qquad V = \prod_{i=0}^{k} \Lambda_i^{a_i},$$

and the new Λ_{k+1} always appears in power one. This yields an alternative method to learn $g_t^{Y_j}$, $j = 1, 2, \ldots$, by the users. By (9),

$$\begin{aligned} V' &= \prod_{i=k'}^{0} g_t^{\tilde{a}_i \cdot \frac{\alpha_1^i + \alpha_2^i}{2}} = g_t^{\frac{1}{2} \sum_{i=k'}^{0} \tilde{a}_i (\alpha_1^i + \alpha_2^i)} \\ &= \left(g_t^{\tilde{f}(\alpha_1) + \tilde{f}(\alpha_2)}\right)^{\frac{1}{2}} = \left(g_t^{\tilde{f}(\alpha_1)} \cdot g_t^{\tilde{f}(\alpha_2)}\right)^{\frac{1}{2}}, \\ V &= \left(g_t^{f(\alpha_1)} \cdot g_t^{f(\alpha_2)}\right)^{\frac{1}{2}}. \end{aligned}$$

Similarly, for

$$Y_j = (\alpha_1^j + \alpha_2^j) \cdot 2^{-1} \bmod q, \tag{17}$$

where $j = 0, 1, \ldots$, we have from (16), (15) and (12) that the direct computation of W_{id} and Q_{id} satisfies the following equations

$$\begin{aligned} W_{id} &= \prod_{j=k-1}^{0} \Lambda_j^{\mu_{id,j}} = g_t^{\frac{1}{2} \sum_{j=k-1}^{0} \mu_{id,j} (\alpha_1^j + \alpha_2^j)} \\ &= \left(g_t^{g_{id}(\alpha_1)} \cdot g_t^{g_{id}(\alpha_2)}\right)^{\frac{1}{2}}, \end{aligned} \tag{18}$$

$$\begin{aligned} Q_{id} &= \prod_{j=k-1}^{0} \Lambda_{j+1}^{\mu_{id,j}} = g_t^{\frac{1}{2} \sum_{j=k-1}^{0} \mu_{id,j} (\alpha_1^{j+1} + \alpha_2^{j+1})} \\ &= \left(g_t^{g_{id}(\alpha_1) \cdot \alpha_1} \cdot g_t^{g_{id}(\alpha_2) \cdot \alpha_2}\right)^{\frac{1}{2}}. \end{aligned} \tag{19}$$

Note that the formulas (18) and (19) are also satisfied for the initial cases: for $k = 0$, i.e. for $R = \emptyset$ (then $g_{id}(X) \equiv 0$), and for $k = 1$, when the first revoked value appears in R (then $g_{id}(X) \equiv 1$).

We shall check that (18) and (19) are invariants of the procedure `UpdateWitness()`, that is, it yields the same results as the direct computations. This means that replacing the vector (10) with the vector (11) cannot be detected by the users. This issue is also crucial for robustness of the `VerifyNonRevocationProof()`: if (in case of the attack) the two methods that could be used by a user would yield different results, then the update made on the side of the Revocation Authority could disagree with the update made by the user, yielding a false verification result.

The case **add**=*true.*

$$\begin{aligned} d'_{id} &:= d_{id} \cdot (x' - x_{id}) = f(-x_{id}) \cdot (-x_{id} + x') = \tilde{f}(-x_{id}). \\ W'_{id} &:= V \cdot W_{id}^{(x' - x_{id})} \\ &= \left(g_t^{f(\alpha_1)} \cdot g_t^{f(\alpha_2)}\right)^{\frac{1}{2}} \cdot \left(g_t^{g_{id}(\alpha_1)} \cdot g_t^{g_{id}(\alpha_2)}\right)^{\frac{1}{2} \cdot (x' - x_{id})} \\ &= \left(g_t^{f(\alpha_1) + g_{id}(\alpha_1) \cdot (x' - x_{id})} \cdot g_t^{f(\alpha_2) + g_{id}(\alpha_2) \cdot (x' - x_{id})}\right)^{\frac{1}{2}}. \end{aligned}$$

Since for "**add**=true" we have:

$$\begin{aligned} f(X) + g_{id}(X) \cdot (x' - x_{id}) &= f(X) + \frac{f(X) - d_{id}}{X + x_{id}} \cdot (x' - x_{id}) \\ &= \frac{f(X) \cdot (X + x_{id}) + (f(X) - d_{id}) \cdot (x' - x_{id})}{(X + x_{id})} \\ &= \frac{f(X) \cdot (X + x_{id}) + f(X) \cdot (x' - x_{id}) - d_{id} \cdot (x' - x_{id})}{(X + x_{id})} \\ &= \frac{f(X) \cdot (X + x') - d'_{id}}{(X + x_{id})} = \frac{\tilde{f}(X) - d'_{id}}{(X + x_{id})} = \tilde{g}_{id}(X), \end{aligned}$$

where the polynomial

$$\tilde{g}_{id}(X) := (\tilde{f}(X) - d'_{id})/(X + x_{id}) = \sum_{j=k'-1}^{0} \tilde{\mu}_{id,j} X^j$$

is the quotient polynomial corresponding to $\tilde{f}(X)$, we have that

$$W'_{id} \;=\; \left(g_t^{\tilde{g}_{id}(\alpha_1)} \cdot g_t^{\tilde{g}_{id}(\alpha_2)}\right)^{\frac{1}{2}} = \prod_{j=k'-1}^{0} \Lambda_j^{\tilde{\mu}_{id,j}}. \quad (20)$$

(the latter equality follows from the equalities (18)). Similarly:

$$\begin{aligned} Q'_{id} \;&:=\; V' \cdot (W'_{id})^{-x_{id}} \cdot g_t^{-d'_{id}} = \left(g_t^{\tilde{f}(\alpha_1)} \cdot g_t^{\tilde{f}(\alpha_2)}\right)^{\frac{1}{2}} \cdot \\ &\qquad \left(\left(g_t^{\tilde{g}_{id}(\alpha_1)} \cdot g_t^{\tilde{g}_{id}(\alpha_2)}\right)^{\frac{1}{2}}\right)^{-x_{id}} \cdot \left(g_t^{-d'_{id}} \cdot g_t^{-d'_{id}}\right)^{\frac{1}{2}} \\ &= \left(g_t^{\tilde{f}(\alpha_1) - x_{id}\cdot\tilde{g}_{id}(\alpha_1) - d'_{id}} \cdot g_t^{\tilde{f}(\alpha_2) - x_{id}\cdot\tilde{g}_{id}(\alpha_2) - d'_{id}}\right)^{\frac{1}{2}}. \end{aligned}$$

Observe that $\tilde{f}(X) - x_{id} \cdot \tilde{g}_{id}(X) - d'_{id}$ can be computed as $\left(\tilde{g}_{id}(X) \cdot (X + x_{id}) + d'_{id}\right) - x_{id} \cdot \tilde{g}_{id}(X) - d'_{id} = \tilde{g}_{id}(X) \cdot X$. Hence

$$Q'_{id} = \left(g_t^{\tilde{g}_{id}(\alpha_1)\cdot\alpha_1} \cdot g_t^{\tilde{g}_{id}(\alpha_2)\cdot\alpha_2}\right)^{\frac{1}{2}} = \prod_{j=k'-1}^{0} \Lambda_{j+1}^{\tilde{\mu}_{id,j}}. \quad (21)$$

The latter equality follows from the equalities (19). From (20) and (21) it follows that in the case "**add**=true" the `UpdateWitness()` procedure and the direct method yield the same results.

The case **add**=*false.* Similarly as before we compute

$$\begin{aligned} d'_{id} &:= d_{id} \cdot (x' - x_{id})^{-1} = f(-x_{id}) \cdot (-x_{id} + x')^{-1} = \tilde{f}(-x_{id}). \\ W'_{id} &:= \left(V'^{-1} \cdot W_{id}\right)^{(x'-x_{id})^{-1}} \\ &= \left(\left(g_t^{\tilde{f}(\alpha_1)} \cdot g_t^{\tilde{f}(\alpha_2)}\right)^{-\frac{1}{2}} \cdot \left(g_t^{g_{id}(\alpha_1)} \cdot g_t^{g_{id}(\alpha_2)}\right)^{\frac{1}{2}}\right)^{(x'-x_{id})^{-1}} \\ &= \left(g_t^{(g_{id}(\alpha_1) - \tilde{f}(\alpha_1))\cdot(x'-x_{id})^{-1}} \cdot g_t^{(g_{id}(\alpha_2) - \tilde{f}(\alpha_2))\cdot(x'-x_{id})^{-1}}\right)^{\frac{1}{2}}. \end{aligned}$$

In the case "**add**=false", x' is removed from R, so

$$\begin{aligned} &\left(g_{id}(X) - \tilde{f}(X)\right) \cdot (x' - x_{id})^{-1} \\ &\quad = (x' - x_{id})^{-1} \cdot \left(g_{id}(X) - \frac{f(X)}{(X + x')}\right) \\ &\quad = (x' - x_{id})^{-1} \cdot \frac{g_{id}(X) \cdot (X + x') - f(X)}{(X + x')} \\ &\quad = (x' - x_{id})^{-1} \cdot \frac{g_{id}(X) \cdot (X + x') - (g_{id}(X) \cdot (X + x_{id}) + d_{id})}{(X + x')} \\ &\quad = (x' - x_{id})^{-1} \cdot \frac{g_{id}(X) \cdot (x' - x_{id}) - d_{id}}{(X + x')} \\ &\quad = \frac{g_{id}(X) - d_{id} \cdot (x' - x_{id})^{-1}}{(X + x')} = \frac{g_{id}(X) - d'_{id}}{(X + x')}. \end{aligned} \quad (22)$$

On the other hand, from the equality

$$\begin{aligned} f(X) \;&=\; g_{id}(X) \cdot (X + x_{id}) + d_{id} \\ &=\; g_{id}(X) \cdot (X + x_{id}) + d'_{id} \cdot (x' - x_{id}) \end{aligned}$$

and the equality

$$\begin{aligned} f(X) \;&=\; \tilde{f}(X) \cdot (X + x') \\ &=\; \left(\tilde{g}_{id}(X) \cdot (X + x_{id}) + d'_{id}\right) \cdot (X + x') \\ &=\; \tilde{g}_{id}(X) \cdot (X + x') \cdot (X + x_{id}) + d'_{id} \cdot (X + x') \end{aligned}$$

it follows that

$$\begin{aligned} &g_{id}(X) \cdot (X + x_{id}) + d'_{id} \cdot (x' - x_{id}) \\ &\quad = \tilde{g}_{id}(X) \cdot (X + x') \cdot (X + x_{id}) + d'_{id} \cdot (X + x'). \end{aligned}$$

The latter equality is equivalent to

$$\begin{aligned} &g_{id}(X) \cdot (X + x_{id}) + d'_{id} \cdot (x' - x_{id}) - d'_{id} \cdot (X + x') \\ &=\; \tilde{g}_{id}(X) \cdot (X + x') \cdot (X + x_{id}), \end{aligned}$$

which in turn is equivalent to the equality

$$g_{id}(X)\cdot(X+x_{id}) - d'_{id}\cdot(X+x_{id}) = \tilde{g}_{id}(X)\cdot(X+x')\cdot(X+x_{id}).$$

Consequently,

$$g_{id}(X) - d'_{id} = \tilde{g}_{id}(X) \cdot (X + x').$$

Hence in (22) $\left(g_{id}(X) - \tilde{f}(X)\right)\cdot(x' - x_{id})^{-1} = \tilde{g}_{id}(X)$. Therefore by (18) we get:

$$W'_{id} = \left(g_t^{\tilde{g}_{id}(\alpha_1)} \cdot g_t^{\tilde{g}_{id}(\alpha_2)}\right)^{\frac{1}{2}} = \prod_{j=k'-1}^{0} \Lambda_j^{\tilde{\mu}_{id,j}}.$$

The reasoning for Q'_{id} is exactly the same as in the case "**add**=true". Accordingly, the `UpdateWitness()` procedure and the direct method again yield the same results.

4.4.3 *Tracing the users*

We assume that the Revocation Authority knows all the values x_{id}.

Since $g^{\delta} = K$ we have $X^{\delta}/Y = W_{id}^{\delta}/Q_{id}$. From (8), (12) and (13) for Y_j defined in (17) we have:

$$\begin{aligned} \frac{W_{id}^{\delta}}{Q_{id}} \;&=\; \frac{W_{id}^{\frac{\alpha_1+\alpha_2}{2}}}{Q_{id}} = \frac{\left(\left(g_t^{g_{id}(\alpha_1)+g_{id}(\alpha_2)}\right)^{\frac{1}{2}}\right)^{\frac{\alpha_1+\alpha_2}{2}}}{\left(g_t^{g_{id}(\alpha_1)\cdot\alpha_1 + g_{id}(\alpha_2)\cdot\alpha_2}\right)^{\frac{1}{2}}} \\ &=\; \left(g_t^{g_{id}(\alpha_1)\cdot\left(\frac{\alpha_1+\alpha_2}{2} - \alpha_1\right) + g_{id}(\alpha_2)\cdot\left(\frac{\alpha_1+\alpha_2}{2} - \alpha_2\right)}\right)^{\frac{1}{2}} \\ &=\; \left(g_t^{g_{id}(\alpha_1)\cdot(\alpha_2-\alpha_1) + g_{id}(\alpha_2)\cdot(\alpha_1-\alpha_2)}\right)^{\frac{1}{4}} \\ &=\; \left(g_t^{g_{id}(\alpha_1) - g_{id}(\alpha_2)}\right)^{\frac{\alpha_2-\alpha_1}{4}} \end{aligned}$$

That is

$$\left(\frac{X^{\delta}}{Y}\right)^{4\cdot(\alpha_2-\alpha_1)^{-1} \bmod q} = g_t^{g_{id}(\alpha_1) - g_{id}(\alpha_2)}. \quad (23)$$

The exponent in the right hand side should depend on x_{id} for polynomial f such that $\deg f \geq 3$ (note that the free coefficient of the polynomial $g_{id}(X)$ disappears in the difference $g_{id}(\alpha_1) - g_{id}(\alpha_2)$, and that $g_{id}(X)$ is monic, so the leading coefficient of $g_{id}(X)$ carries no information about x_{id}). Hence in the worst case the Revocation Authority may collude with three users to ensure that $\deg f \geq 3$.

Similarly like in Subsect.4.3 tracing is possible even if Y is not contained in the output of the non-revocation proof. We may use formula (23) to design an alternative version of the attack.

5. FINAL REMARKS

The list of problems presented in this paper is not complete as we are aware of more threats. In each case, one can extend the scheme so that a particular attack does not work. However, addtitional procedures make the system even more "heavy" and can make room

for new trapdoors. Definitely, since U-Prove is a proprietary system, this is the role of its designers decide how to patch the system.

In our attacks the Revocation Authority learns the revocation attribute when asked for checking X and Y. Interestingly, for [8] the non-revocation proof is checked by the verifier himself and all attack scenarios from this paper do not apply.

A general suggestion is to separate strictly the verifier and the Revocation Authority and blind the parameters X, Y before passing them to the Revocation Authority. That is, instead of X, Y the Revocation Authority should get $X' = X^t, Y' = Y^t$ for a random t. Of course, this does not secure the user against collusion between the verifier and the Revocation Authority, as the verifier must use the original X, Y as arguments for the hash function $\mathcal{H}$. So again, the user has to trust blindly the verifier.

6. REFERENCES

[1] Security Team, IBM Research Zurich: Specification of the Identity Mixer Cryptographic Library. IBM Research Report 3730, IBM Research (2010). Available from: http://domino.research.ibm.com/library/cyberdig.nsf/papers/EEB54FF3B91C1D648525759B004FBBB1/$File/rz3730_revised.pdf

[2] Paquin, C.: On the Revocation of U-Prove Tokens. Tech. Report, Microsoft Research (September 2014). Available from: http://research.microsoft.com/pubs/228729/On%20the%20revocation%20of%20U-Prove%20tokens.pdf

[3] Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Pfitzmann, B., ed.: EUROCRYPT. Vol. 2045 of LNCS, Springer (2001) 93–118

[4] Paquin, C., Zaverucha, G.: U-Prove cryptographic specification v1.1 (revision 3). Technical report (December 2013). http://research.microsoft.com/apps/pubs/default.aspx?id=166969

[5] Brands, S.: Untraceable off-line cash in wallets with observers (extended abstract). In Stinson, D.R., ed.: CRYPTO. Volume 773 of LNCS, Springer (1993) 302–318

[6] Brands, S.A.: Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy. 1 edn. MIT Press, Cambridge-London (2000). http://www.credentica.com/the_mit_pressbook.html

[7] Microsoft: U-Prove. Webpage of the project (retrieved 2014). http://research.microsoft.com/en-us/projects/u-prove/

[8] Acar, T., Chow, S.S.M., Nguyen, L.: Accumulators and U-Prove revocation. In Sadeghi, A.R., ed.: Financial Cryptography. Vol. 7859 of LNCS, Springer (2013) 189–196

[9] Nguyen, L., Paquin, C.: U-Prove designated-verifier accumulator revocation extension. Technical Report Draft Revision 1, Microsoft Research (September 2013 (updated February 2014)). http://research.microsoft.com/pubs/200817

[10] Nguyen, L., Paquin, C.: U-Prove designated-verifier accumulator revocation extension. Technical Report Draft Revision 2, Microsoft Research (July 2014). http://research.microsoft.com/apps/pubs/default.aspx?id=219671

[11] Hanzlik, L., Kluczniak, K., Kutyłowski, M.: Attack on a U-Prove revocation scheme. In: Financial Cryptography'2014, to appear in LNCS

[12] ABC4TRUST. Webpage of the project (retrieved 2014). https://abc4trust.eu/

[13] Dahlquist, G., Åke Björck: Numerical Methods in Scientific Computing: Volume 1. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2008)

[14] Shoup, V.: NTL: A library for doing number theory. http://www.shoup.net/ntl/ (2014) Ver.6.1.

[15] European Network of Excellence in Cryptology II: Main computational assumptions in cryptography. Deliverable 3 of the virtual lab MAYA (D.MAYA.3) (2010)

APPENDIX

A. EQUIVALENCE OF THE WAYS OF COMPUTING T_1, T_2, T_3

In this section we show that the ways of computing T_1, T_2 and T_3 by the procedures `GenerateNonRevocationProof()` and `VerifyNonRevocationProof()` are equivalent. The main point is that this holds even if the parameter d is manipulated and not computed according to `ComputeWitness()`. Otherwise, we assume that the remaining parameters are computed as described in Sect. 2.

First we check two ways of computing T_1. According to `GenerateNonRevocationProof()`

$$T_1 = X^{k_1}(\tilde{c}_{id}K)^{-k_2}g_1^{k_3}$$

while `VerifyNonRevocationProof()` states that

$$T_1 = (VY^{-1}(C_d)^{-1})^{c'}X^{s_1}(\tilde{c}_{id}K)^{-s_2}g_1^{s_3}$$

Hence we have to prove that

$$(VY^{-1}(C_d)^{-1})^{c'}X^{s_1}(\tilde{c}_{id}K)^{-s_2}g_1^{s_3} \stackrel{?}{=} X^{k_1}(\tilde{c}_{id}K)^{-k_2}g_1^{k_3}$$

Below we get a sequence of equations equivalent to the inspected one:

$$(VY^{-1}(C_d)^{-1})^{c'}X^{s_1-k_1}(\tilde{c}_{id}K)^{-s_2+k_2}g_1^{s_3-k_3} \stackrel{?}{=} 1$$
$$(VY^{-1}(C_d)^{-1})^{c'}X^{-c'x_{id}}(\tilde{c}_{id}K)^{c't_1}g_1^{-c'z} \stackrel{?}{=} 1$$
$$VY^{-1}(C_d)^{-1}X^{-x_{id}}(\tilde{c}_{id}K)^{t_1}g_1^{-z} \stackrel{?}{=} 1$$
$$V(QK^{t_1})^{-1}(C_d)^{-1}X^{-x_{id}}(\tilde{c}_{id}K)^{t_1}g_1^{-z} \stackrel{?}{=} 1$$
$$VQ^{-1}(C_d)^{-1}X^{-x_{id}}(\tilde{c}_{id})^{t_1}g_1^{-z} \stackrel{?}{=} 1$$
$$V(VW^{-x_{id}}g_t^{-d})^{-1}(C_d)^{-1}X^{-x_{id}}(\tilde{c}_{id})^{t_1}g_1^{-z} \stackrel{?}{=} 1$$
$$W^{x_{id}}g_t^d(C_d)^{-1}X^{-x_{id}}(\tilde{c}_{id})^{t_1}g_1^{-z} \stackrel{?}{=} 1$$
$$W^{x_{id}}g_t^d(C_d)^{-1}(Wg^{t_1})^{-x_{id}}(\tilde{c}_{id})^{t_1}g_1^{-z} \stackrel{?}{=} 1$$
$$g_t^d(C_d)^{-1}(g^{t_1})^{-x_{id}}(\tilde{c}_{id})^{t_1}g_1^{-z} \stackrel{?}{=} 1$$
$$g_t^d(g_t^dg_1^{t_2})^{-1}(g^{t_1})^{-x_{id}}(\tilde{c}_{id})^{t_1}g_1^{-z} \stackrel{?}{=} 1$$
$$g_1^{-t_2}(g^{t_1})^{-x_{id}}(\tilde{c}_{id})^{t_1}g_1^{-z} \stackrel{?}{=} 1$$
$$g_1^{-t_2}(g^{t_1})^{-x_{id}}(\tilde{c}_{id})^{t_1}g_1^{-(t_1\tilde{o}_{id}-t_2)} \stackrel{?}{=} 1$$
$$(g^{t_1})^{-x_{id}}(\tilde{c}_{id})^{t_1}g_1^{-t_1\tilde{o}_{id}} \stackrel{?}{=} 1$$
$$(g^{t_1})^{-x_{id}}(g^{x_{id}}g_1^{\tilde{o}_{id}})^{t_1}g_1^{-t_1\tilde{o}_{id}} \stackrel{?}{=} 1$$

The last equation is obviously true.

Similarly, we have to check equivalence of two ways of computing T_2, which is once computed as $\tilde{c}_{id}^{c'}g^{s_1}g_1^{s_4}$ and once as $g^{k_1}g_1^{k_4}$. The following equations are equivalent:

$$g^{k_1}g_1^{k_4} \stackrel{?}{=} \tilde{c}_{id}^{c'}g^{s_1}g_1^{s_4}$$
$$1 \stackrel{?}{=} \tilde{c}_{id}^{c'}g^{s_1-k_1}g_1^{s_4-k_4}$$
$$1 \stackrel{?}{=} \tilde{c}_{id}^{c'}g^{-c'x_{id}}g_1^{-c'\tilde{o}_{id}}$$
$$1 \stackrel{?}{=} (g^{x_{id}}g_1^{\tilde{o}_{id}})^{c'}g^{-c'x_{id}}g_1^{-c'\tilde{o}_{id}}$$

The last equation is obviously true.

Finally, we check equivalence of two ways of computing T_3:

$$C_d^{k_5}g_1^{k_6} \stackrel{?}{=} g_t^{c'}(C_d)^{s_5}g_1^{s_6}$$
$$1 \stackrel{?}{=} g_t^{c'}(C_d)^{s_5-k_5}g_1^{s_6-k_6}$$
$$1 \stackrel{?}{=} g_t^{c'}(C_d)^{-c'w}g_1^{-c'z'}$$
$$1 \stackrel{?}{=} g_tC_d^{-w}g_1^{-z'}$$
$$1 \stackrel{?}{=} g_t(g_t^dg_1^{t_2})^{-w}g_1^{-z'}$$
$$1 \stackrel{?}{=} g_t^{1-dw}(g_1^{t_2})^{-w}g_1^{-z'}$$
$$1 \stackrel{?}{=} (g_1^{t_2})^{-w}g_1^{-z'}$$
$$1 \stackrel{?}{=} (g_1^{-t_2w-z'})$$
$$1 \stackrel{?}{=} g_1^0$$