Untraceable Uniqueness-Proving with Multi-Context One-Show Credentials

Jeffrey Lim

May 20, 2015

6.UAP Final Report

1 Overview

In this paper I examine an approach towards *multi-context one-show* (MCOS) credentials, which enable a user to demonstrate possession of a credential only once within a given context, but arbitrarily many times across different contexts.

The prototypical use case for MCOS credentials is to prevent double-voting in polls. Suppose that a credential-issuing organization is trusted to issue credentials to users in a one-to-one correspondence with their real-world identities (e.g. by checking their drivers licenses). Then, a user, having obtained such a credential, should be able to prove their uniqueness as a voter in any number of polls vis-à-vis the other voters in the poll.

Additionally, it is desirable that the users enjoy a certain degree of privacy. Even if the credential-issuer colludes with the poll-runners, it should be impossible for any of them to link the user's vote to their real-world identity stored with the issuer ("anonymity"), or to link any two of their votes to each other ("unlinkability").

I propose here a MCOS protocol based on the cut-and-choose method described by Chaum, Fiat, and Naor [4], and the parallelized Schnorr protocol for zero-knowledge proofs of knowledge of discrete logarithms, as described by Camenisch and Michels [2]. After exploring its security properties, I test its performance by giving a minimal implementation in JavaScript. I conclude with a discussion of possible future work and applications.

2 Background

Chaum [3] introduces the notion of *blind signatures*, which allow for a signing authority to issue signatures without knowing what they are signing. In Chaum's scenario, a customer deposits funds at a bank and presents a blinded signature request, which the bank signs. The customer unblinds the signature to produce a signed "bill," waits some time, and gives it to a merchant, who presents it to the bank for redemption. The bank can validate the signature as having been legitimately issued by the bank, but cannot connect it to the customer's original deposit—they know only that they issued the bill to *some* customer.

This is an example of a one-show credential—the bank can record the ID of the bill so that they know not to accept it again. It should not be possible for the customer to mutate the bill so that it has a different ID but still a valid signature. Thus, the only way to adapt Chaum's system for the MCOS case would be to use a different signing key for each poll; this would require that the credential-issuer know in advance the set of polls that the users will vote in.

Camenisch and Lysyanskaya [1] give a protocol for unlinkable multi-show credentials, whereby a user can demonstrate possession of a credential by a zero-knowledge proof that does not allow different uses of the credential to be connected. This has the advantage that the user can obtain their credential once and for all, and never again have to interact with the issuer. However, this same property makes Camenisch and Lysyanskaya's scheme unsuitable for MCOS, since there is no way of preventing a user from voting an unlimited number of times.

The protocol I propose combines elements of both schemes to achieve the goal of MCOS credentials. One caveat applies: It will be necessary for the user to obtain in advance some number of "voter-tickets" from the credential issuer, each of which will be used once and then discarded. While this is inconvenient, it is not as much so as in Chaum's system, for two

reasons. First, the user may obtain a large batch of tickets simultaneously and they will still not be linkable to each other; and second, the issuer does not need to know anything about the polls that the user will eventually vote in.

3 Description of protocol

We describe the interactions between a credential issuer ("the Issuer"), which enables a user "Alice" to vote in a poll operated by "Bob."

3.1 Setup

The Issuer selects an RSA public modulus/exponent pair (n, e), and a prime modulus and generator (p, g), whose order is some prime q, and publishes these values. They keep their RSA private exponent d secret. All parties agree on some cryptographic hash function H, which takes any number of arguments. Two security parameters, k and m, are agreed upon.

3.2 Registering identity

Alice selects her secret $x \in_R \mathbb{Z}_p^*$ and finds $y = g^x \bmod p$. She sends y to the Issuer, along with her external credentials (drivers license, etc.). If the Issuer is satisfied that Alice is not already in their database, they add her information and y value to their database.

3.3 Obtaining voter-tickets

Before she votes in any polls, Alice must first obtain one or more *voter-tickets*, each of which will be used exactly once (one ticket for each different poll) before being discarded. However, it is not necessary for Alice to know at this time which or how many polls she will eventually vote in; furthermore, she can obtain these tickets in large batches at any time without compromising anonymity or unlinkability. The following illustrates the process for Alice to obtain one voter-ticket.

1. Alice selects random $r_1, r_2, \ldots r_k \in_R \mathbb{Z}_q^*$ and $b_1, b_2, \ldots b_k \in_R \mathbb{Z}_n$ (with the b values being relatively prime to n.)
2. For $1 \leq i \leq k$, she finds $h_i = {b_i}^e \cdot H(y^{r_i}, g^{r_i}) \bmod n$.
3. She sends all k of the h_i values to the Issuer, along with y.
4. The Issuer verifies that y is in their database, and chooses a random subset $C \subset \{1, 2, \ldots k\}$, with $\|C\| = \frac{k}{2}$, and sends C to Alice. (Let $\bar{C}$ denote the complement of this set; i.e. $[1, k] - C$.)
5. Alice replies with (r_i, b_i) for all i in C, and the Issuer verifies that all of the corresponding h_i values fit the form specified above. ("Cut-and-Choose")
6. The Issuer replies with $S' = \prod_{j \in \bar{C}} h_j^d \bmod n$.
7. Alice finds $S = S' \cdot \prod_{j \in \bar{C}} b_j^{-1} \bmod n$. This S, along with the list of the remaining (y^{r_j}, g^{r_j}) pairs, comprises the voter-ticket.

In practice, Alice can significantly decrease the storage requirements by choosing $r_1, r_2, \ldots r_k$ as the deterministic output of some pseudorandom function starting from some master seed R; e.g. $r_1 = H(R, 1), r_2 = H(R, 2)$, etc. Then, she only needs to store $(S, R, \bar{C})$, which is enough for her to reconstruct the complete voter-ticket when it is needed.

3.4 Voting

Bob chooses and publishes a generator f (also of order q) with respect to the modulus p, which should be sufficiently random as to be unique to this particular poll, with high probability. For example, it could be produced by some pseudorandom function of Bob's domain name, timestamp, etc.[1] This f is known as the "poll base."

Alice finds $z = f^x \bmod p$ and sends it to Bob, who checks that this z is not already in his database. (If it is, that means that Alice has voted in this poll already.) Then, Alice sends Bob a complete voter-ticket (y^{r_j}, g^{r_j}) for all $j \in \bar{C}$, along with the signature S, which Bob verifies by finding $\prod_{j \in \bar{C}} H(y^{r_j}, g^{r_j}) \bmod n$ and checking that this is equal to $S^e \bmod n$.[2]

Finally, Alice uses the parallelized Schnorr protocol (described below) to prove to Bob the following claim:

> "I know α such that $y^{r_j} = (g^{r_j})^\alpha \bmod p$, and... [likewise for all $j \in \bar{C}$] ... and $z = f^\alpha \bmod p$."

[1] This can be found, for example, as $f = r^{(p-1)/q} \bmod p$, for some random $r \in_R \mathbb{Z}_p^*$.

[2] The use of j and $\bar{C}$ in this section is merely a notational convention; Alice should not disclose these values to Bob, since otherwise he could almost certainly link the vote back to Alice's real-world identity, were he to collude with the Issuer.

In fact, this claim is satisfied by $\alpha = x$, but Alice does not reveal x in carrying out the proof.

If Bob is satisfied with the proof, he records Alice's vote and adds z to his database. Alice can now discard the voter-ticket (since, if she were to use it again, even in a different poll, it would be linkable to the present vote with Bob).

4 Parallelized Schnorr ZKP protocol

Schnorr [7] gives a protocol for constructing zero-knowledge proofs of claims of the form "I know α such that $y = g^\alpha \bmod p$," where g is a generator of prime order q in the field modulo a prime p, and y is some public value. (In general, Roman letters denote values known to both the prover and the verifier, while Greek letters denote values whose knowledge is to be zero-knowledge-proved.) Camenisch and Michels [2] extend this protocol to conjunctions of such statements over the same α; i.e. "I know α such that $(y_1 = g_1^\alpha) \wedge (y_2 = g_2^\alpha) \wedge \ldots \wedge (y_k = g_k^\alpha) \bmod p$." We will describe Camenisch and Michels's extension ("parallelized Schnorr"), and briefly analyze its security.

4.1 Description

Alice wishes to prove to Bob that she knows α, as stated above. They perform the following steps:

1. Alice chooses random $r \in_R \mathbb{Z}_p^*$, and sends $t_i = g_i^r$ to Bob, for $i \in [1, k]$.
2. Bob chooses random $c \in_R [0, 2^m - 1]$ and sends it to Alice (where m is a security parameter).
3. Alice responds with $s = r - c\alpha$.
4. Bob accepts the proof if $t_i = g_i^s y_i^c$ for all $i \in [1, k]$.

We call this procedure "parallelized Schnorr" because it is essentially executing Schnorr's protocol in parallel for each (y_i, g_i) pair simultaneously, using the same r, s, c for each.

Assuming that H is a good hash function, we can make this procedure non-interactive by a variant of the Fiat-Shamir heuristic [5]. Instead of obtaining c from Bob, Alice calculates $c = H(t_1, t_2, \ldots t_k) \bmod 2^m$. Then, since this c is not predictable while Alice is computing her commitments, the probative effect of the interaction is the same as if Bob had chosen c randomly himself. In this way Alice can perform the entire proof in a single message.

4.2 Analysis

In analyzing this protocol, we make use of Schnorr's result, which we will not prove here:

> In the ordinary case of the (non-parallelized) Schnorr protocol with $k = 1$, no polynomial-time adversary that does not actually know α can forge an acceptable proof with probability more than 2^{-m}, assuming that finding discrete logarithms is hard.

From this it follows that the same probability bound applies to the $k > 1$ case as well. Suppose, without loss of generality, that the best α that Alice knows is one that satisfies all of the relations but one, so $y_i = g_i^\alpha \bmod p$ for all i except $i = 1$. Then suppose that Bob executes a "forgiving" modification of parallelized Schnorr, wherein he simply ignores the verification results $t_i \stackrel{?}{=} g_i^s y_i^c$ for $i \neq 1$, and decides to accept or reject solely on the basis of $t_1 \stackrel{?}{=} g_1^s y_1^c$. This "forgiving" verification method is in fact identical to ordinary (non-parallelized) Schnorr, applied to (y_1, g_1). But, since actual parallelized Schnorr returns "reject" in strictly more cases, the probability of deceiving it must be no greater than for ordinary Schnorr.

5 Security

We now show that the protocol presented above has all three desired security properties, under the discrete logarithm, decisional Diffie-Hellman, and RSA computational hardness assumptions, and assuming that the parallelized Schnorr protocol is in fact sound and zero-knowledge. We also assume that the Issuer is trusted not to issue signatures except according to the protocol (since otherwise, preventing double-voting is clearly futile).

However, we otherwise permit that all parties may be malicious, and that any subset thereof may collude against the interests of any other. Lastly, we assume that there is more than one user, so that the notions of anonymity and unlinkability are meaningful.

5.1 No double-voting

Alice can double-vote if she can trick the Issuer into signing a fake voter-ticket comprised of entries $(g^{r_j}, \hat{y}^{r_j})$, where $\hat{y} = g^{\hat{x}}$ and $\hat{x} \neq x$ is some secret value known to Alice. (Assuming that H is one-way and that RSA is secure, Alice cannot outright forge these signatures.) Then, she would be able to perform the ZKP with both $z = f^x$ and $\hat{z} = f^{\hat{x}}$; and since $z \neq \hat{z}$, Bob will consider these two distinct votes.

However, Alice can only do this if, during the Cut-and-Choose phase of the protocol, she constructs exactly $\frac{k}{2}$ of the entries according to the fake form with $\hat{x}$, and these just so happen to be exactly the entries that the Issuer *does not* request Alice to reveal. The probability of this happening is $\binom{k}{k/2}^{-1}$, which is $O(2^{-k/2})$, small enough to ignore, especially if the Issuer rate-limits Alice after a certain number of failed challenges.[3]

Otherwise, the only way that Alice can double-vote is if she can find some $\hat{x}$ such that $y^{r_j} = (g^{r_j})^{\hat{x}} \bmod p$ for all entries in the ticket, but where $f^{\hat{x}} \neq f^x \bmod p$. This is impossible so long as g and f are both generators of the same order q.

5.2 Anonymity

The blinding scheme used in the voter-ticket issuance phase (multiplying the hashes by b_i^e, and dividing the signature by $\prod b_i$) ensures that the Issuer cannot link the issuance of a ticket with that same ticket when it is used. In other words, even if the Issuer records all interactions and colludes with Bob when Alice displays her ticket, they cannot determine which of the identities in their database the ticket corresponds to.

Each blinding factor b is chosen uniformly at random over integers relatively prime to n; and since the RSA signing/verification functions are bijective, the b^e values also have the same distribution. This means that, upon seeing $b^e h$, the Issuer learns nothing about h other than that it differs multiplicatively from $b^e h$ by some factor that is relatively prime to n; but this only narrows the possibility set by a factor of $\phi(n)/n$, which is negligibly less than 1 (i.e. it differs from 1 by a factor that shrinks exponentially in $|n|$).

[3]Even this concern may be obviated by making k large enough; however, since modular exponentiation is rather expensive, it is preferable to keep k as small as possible.

5.3 Unlinkability

To ensure unlinkability, Alice should use a different voter-ticket every time she votes. From a single ticket it is impossible to tell what her private value x is, since this is an instance of the discrete logarithm problem. As well, given two entries (y_1, g_1) and (y_2, g_2) from two different tickets, the problem of determining whether the tickets were created using the same user secret x is at least as hard as the decisional Diffie-Hellman problem—if an algorithm $A(y_1, g_1, y_2, g_2)$ could tell us whether there exists some α such that $y_1 = g_1^\alpha$ and $y_2 = g_2^\alpha \bmod p$, then $A(g, g^a, g^b, g^{ab}) \wedge A(g, g^b, g^a, g^{ab})$ would tell us whether (g^a, g^b, g^{ab}) is a Diffie-Hellman triplet.

The same is true of the problem of determining, given two different poll bases f_1 and f_2, whether f_1^x and f_2^x were created using the same x value. An adversary can do this only if they already know the discrete logarithm of f_1 to the base f_2 (or vice-versa); if both are chosen by a verifiable random method, then this would itself be another discrete logarithm problem.

Lastly, under the assumption that the parallelized Schnorr protocol is indeed zero-knowledge, there is no way for an adversary to tell whether two instances of the proof using separate tickets were generated using the same x value.

6 Implementation

For a reasonable security level, we set $k = 60$, $m = 80$, and the RSA key size $|n| = 2048$. We set the prime modulus and group order sizes $|p| = 2048$ and $|q| = 256$, following NIST recommendations for DSA keys[4]. We use SHA-256 as our hash function H, on the assumption that it has the desired pseudorandom and one-way properties. We then seek to determine whether typical computer systems can execute the protocol in a reasonable amount of time.

6.1 Setup

All functions are implemented in JavaScript, using the Stanford JavaScript Crypto Library (SJCL)[5] and JavaScript Big Number (JSBN)[6] as dependencies. The code can be run in either a client (web browser) or server (NodeJS) environment.

[4]`http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf`

[5]`https://github.com/bitwiseshiftleft/sjcl`

[6]`http://www-cs-students.stanford.edu/~tjw/jsbn/`

We run two benchmark tests to time the execution of the protocol: issuing a voter-ticket (§3.3), and presenting the ticket (§3.4). These tests simulate the roles of all three parties one after another—there is no network communication or parallelization. The tests are repeated 20 times on a freshly-started system with no other processes running; we measure the duration (in milliseconds) of each round of testing.

The tests are run in the following environments:

1. MacBook Air 11-inch, OS X Yosemite 10.10.2, 1.6 GHz Intel Core i5 processor; 4 GB 1600 MHz DDR3 memory. In browser: Google Chrome 43.0.2357.65 (64-bit)
2. Same system. In browser: Firefox 38.0.1
3. Same system. In browser: Tor Browser 4.5.1
4. Same system. In terminal: NodeJS v0.12.3

The code may be accessed here: `https://github.com/jatchili/mcos`.

6.2 Results

The results are given in milliseconds:

Env.	Issuing a ticket		Presenting a ticket	
	Mean	St.Dev.	Mean	St.Dev.
1	3680.7	119.7	14532.8	67.6
2	2962.6	53.7	11674.3	48.2
3	2977.7	39.5	12226.4	199.1
4	3783.3	80.6	20772.7	3005.0

6.3 Analysis

These observations are pessimistic, mainly because the tests do not take advantage of parallelization. If a client and server engage in several sequential rounds of the protocol (e.g. to issue a batch of voter-tickets, or to submit votes in several different polls), then the time can be approximately halved by having the client prepare the next request while the server is verifying the last.

These results do show that implementing this protocol is well within the realm of feasibility for ordinary computer systems available today. However, in the context of web browsers, extremely liberal use of MCOS credentials—such as issuing 1,000 voter-tickets simultaneously, or using a separate ticket for every single "like" or "upvote" action on a website—is likely to be inconvenient unless one can access faster cryptographic primitives than those available in JavaScript. A more practical approach would be to use one ticket per session or user account.

6.4 Other desirable features

Several additional features would be desirable in a larger-scale implementation, but are not implemented here.

- **Bloom filter for double-vote-checking**: Rather than searching linearly for Alice's z in his database, Bob can quickly confirm z's absence by using a Bloom filter. This would be helpful if there are a large number of voters in the same poll.
- **Parameter verification**: We do not check whether the fixed parameters n, e, p, q, g, f are correctly chosen according to the protocol. (This would only need to be done once.)
- **Authentication of voter-ticket requests**: Strictly speaking it is unnecessary for the Issuer to authenticate the origin of voter-ticket requests—Alice may obtain tickets created with some other user's y value, but they would be useless to her, since she would not know the corresponding x. However, to discourage denial-of-service attacks, it may be desirable for the Issuer to rate-limit these requests on a per-user basis.

7 Conclusion

7.1 Future work

A key question is whether it is possible to eliminate the need for voter-tickets, and instead issue credentials that can be used an (effectively) unlimited number of times. Certainly, it would be desirable if the role of the Issuer could be reduced to the point where they do not need to store their private key on any online machines, and instead only use it during the manual verification phase (§3.2). One possible way to do this would be to have the Issuer blindly sign the user's secret x, returning a signature s; Alice would then find $z = H(f, x)$ and perform to Bob a zero-knowledge proof of the claim "I know (α, σ) such that $z = H(f, \alpha)$ and σ is a valid signature of α."

However, we could then no longer use an algebraic zero-knowledge proof technique; we would instead have to turn to fully general methods for zero-knowledge-proving knowledge of a satisfying assignment of an arbitrary Boolean circuit. Jawurek, Kerschbaum, and Orlandi [6] give a promising approach that can be used on a SHA-256 or AES circuit; however, when combined with signature verification, the resulting circuit is prohibitively large for practical use (although still polynomial-sized). An efficient method for performing such a proof would make it possible to base a protocol off of this technique.

7.2 Applications

The system presented here has a wide variety of immediate applications beyond simply polling. In general, any setting in which persons interact anonymously over the Internet leaves open the possibility of a "Sibyl attack"—a large number of identities that are in fact controlled by the same person. MCOS credentials could help alleviate these concerns without requiring that users disclose their real identities to everyone they interact with.

For example, MCOS credentials could be used as an alternative to CAPTCHAs in preventing excessive use of free services. A commerce platform could use them to help ensure that reviews and endorsements are genuine. In general, the possession of an identity that cannot simply be abandoned and restarted at will could serve to incentivize good behavior, without any loss of privacy.

References

[1] Camenisch, Jan, and Anna Lysyanskaya. "An efficient system for non-transferable anonymous credentials with optional anonymity revocation." *EUROCRYPT '01* (2001): 93-118.

[2] Camenisch, Jan, and Markus Michels. "Separability and Efficiency for Generic Group Signature Schemes." *Advances in Cryptology – CRYPTO '99 Lecture Notes in Computer Science* (1999): 413-30.

[3] Chaum, David. "Blind Signatures for Untraceable Payments." *Advances in Cryptology* (1983): 199-203.

[4] Chaum, David, Amos Fiat, and Moni Naor. "Untraceable Electronic Cash." *Advances in Cryptology – CRYPTO '88 Lecture Notes in Computer Science* (1990): 319-27.

[5] Fiat, Amos, and Adi Shamir. "How To Prove Yourself: Practical Solutions to Identification and Signature Problems." *Advances in Cryptology – CRYPTO '86 Lecture Notes in Computer Science* (1986): 186-94.

[6] Jawurek, Marek, Florian Kerschbaum, and Claudio Orlandi. "Zero-knowledge Using Garbled Circuits." *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security – CCS '13* (2013).

[7] Schnorr, C.P. "Efficient Identification and Signatures for Smart Cards." *Advances in Cryptology – CRYPTO '89 Lecture Notes in Computer Science* (1989): 239-52.