Accumulators and U-Prove Revocation*

Tolga Acar[1], Sherman S.M. Chow[2], and Lan Nguyen[3]

[1] Intel Corporation
tolga.acar@intel.com
[2] Microsoft Research
lan.duy.nguyen@microsoft.com
[3] Department of Information Engineering
Chinese University of Hong Kong
sherman@ie.cuhk.edu.hk

Abstract. This work introduces the most efficient universal accumulator known today. For the first time, we have an accumulator which does not depend on hidden order groups, does not require any exponentiations in the target group associated with the pairing function, and only requires two pairings to verify a proof-of-knowledge of a witness.
We present implementations of our accumulator and another recent proposal utilizing Groth-Sahai proofs, with performance results. Our implementations are designed with cryptography agility in mind. We then built a library for revoking anonymous credentials using any accumulators, and integrated it with Microsoft U-Prove, which has a significant contribution to an European Union's privacy standardization effort. Our work enables U-Prove revocation without compromising untraceability.

Keywords: dynamic universal accumulator, U-Prove, revocation, blacklist, privacy enhancing technologies, efficiency, anonymity, accountability, authentication, pairing, implementation, cryptography agility

1 Introduction

Accumulator. A cryptographic accumulator allows aggregation of a large set of elements into one constant-size *accumulator value*, and proving about whether an element has been accumulated. Via the proof system, a prover with a *witness* can convince a verifier about the truth of a *statement*, but any adversary cannot convince a verifier about a false statement. The basic proof is about *membership* statement, for proving that an element has been accumulated. An accumulator is said to be *universal* if it has another proof system, called *non-membership*, to prove that a given element is not accumulated. Moreover, if the costs of updating the accumulator or witnesses, when elements are added to or deleted from the accumulator, do not depend on the number of elements aggregated, we say it is *dynamic*. Two universal dynamic accumulators have been proposed so far. One is based on the Strong RSA assumption in hidden order groups [16], which

* This is an extended abstract. We thank the help of Benoît Libert and Toru Nakanishi.

is derived from a basic (non-universal) accumulator proposed [8]. The other is based on the Strong Diffie-Hellman assumption in prime-order groups with bilinear map (or pairing) [3] which is derived from a pairing-based non-universal accumulator proposed [19]. There is another pairing-based dynamic accumulator [7]; however, it is not universal.

Applications. Accumulators have been used in various applications for different purposes. Two of its major benefits include minimizing the bandwidth requirement and protecting privacy. With the proliferation of mobile devices, bandwidth requirement is becoming more crucial. On the other hand, privacy is a growing concern in various different sectors such as healthcare, military, intelligence, and mobile devices industry. The emergence of cloud computing also leads to a different balance between trust and anonymity in identity management. Some applications of accumulators include space-efficient time-stamping [4], electronic voting [10], and many privacy-preserving authentication mechanisms which include ad-hoc anonymous authentication [12], (ID-based) ring signatures [12, 19, 11], dynamic k-times anonymous authentication [2], and *anonymous credentials* which are *revokable* [8, 7].

Anonymous Credentials. Using *anonymous credentials* a user can prove the possession of some credentials without revealing any other private information such as her identity. Applications include direct anonymous attestation [6], anonymous electronic identity token [9, 17], and implementations such as U-Prove [17], Idemix [9] and in Java cards [5]. In practice, *revocation* is indispensable in credential systems, as dispute, compromise, mistake, identity change, hacking and insecurity could make any credential become invalid before its expiration, especially when they are carried around by mobile devices.

Revocation. Revoking credentials is a notorious issue in cryptography, not to say anonymous credentials. Consider public key infrastructure, there is a certificate revocation list which consists of invalid certificates, and is time-stamped and signed by a certificate authority. Checking if a certificate is revoked requires searching for the certificate's identity in the entire list. For anonymous credential, it is even trickier. In particular, the anonymity and the run-time requirements of honest users should not be affected by revocation of other credentials.

Our contribution. The major contribution of this work is our efficiency improvement of (non-)membership proofs associated with an accumulator, which brings benefits to many existing applications utilizing accumulator. Our efficiency gain comes from a new design which only requires proving equations in a base group $\mathbb{G}_1$ associated with the curves featuring the pairing function, instead of the standard approach in the literature which proves about some pairing equations. As a result, not only the number of pairing operations is greatly reduced, there are also no exponentiation or other operations in the target group $\mathbb{G}_T$ (nor in the other base group $\mathbb{G}_2$), which are often inefficient when compared with the operations in $\mathbb{G}_1$. (Exponentiation in $\mathbb{G}_T$ could be 4 or 5 times more expensive than that in $\mathbb{G}_1$, and exponentiation in $\mathbb{G}_2$ could still be 1.5 to 2 times more expensive.) In particular, our proof systems require less exponentiations and pairings compared to previous pairing-based schemes [19, 3] which use sym-

metric pairings. It is especially suitable for Barreto-Naehrig pairing curves (BN curves) [14], arguably one of the most efficient pairing-friendly curves, where $\mathbb{G}_1$'s operations are also very efficient.

2 U-Prove with Revocation using Our New Accumulator

In this section, we show how to use the non-membership proof of our accumulator to blacklist U-Prove tokens. It will be embedded in the U-Prove presentation protocol as detailed in U-Prove Crypto Specification V1.1 ('Spec') [17]. Due to the lack of spaces, this part is a sketch and should be read in conjunction with the Spec [17]. The text "[**For Revocation**]" highlights the additions of revocation parts to the existing U-Prove protocols.

2.1 Entities

In the original U-Prove Spec, there are Issuer, Prover and Verifier. Issuer issues each Prover U-Prove tokens that contain Issuer's certification of the Prover's attributes. A Prover can prove some of her attributes to a Verifier in a U-Prove token presentation protocol. Besides these three entities as in Spec, we introduce *Blacklist Authority (BA)*, who could be the same as or independent from Issuer. There could be several BAs and several blacklists. Apart from proving Prover's attributes, our new token presentation protocol can also prove that the token is not in any of the blacklists, with full anonymity.

2.2 System Parameters

The generation of the system parameters follows closely as the original Spec, except that the system parameters for the accumulator will also be included.

Issuer parameters: Define

$$IP := (\mathsf{UID}_\mathrm{P},\ \mathsf{desc}(\mathbb{G}_q),\ \mathsf{UID}_\mathcal{H}, (g_0, g_1, \ldots,\ g_n,\ g_t), (\mathrm{e}_1, \ldots,\ \mathrm{e}_n), S))$$

where $\mathsf{UID}_\mathcal{H}$ is an application-specific unique identifier for Issuer parameters and $\mathsf{UID}_\mathcal{H}$ is an identifier of a cryptographically secure hash algorithm.

[**For Revocation**] **Accumulator parameters:** Define

$$\mathsf{param} := (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P_1, P_2, P_{pub}, H, K, G_1)$$

where $P_{pub} := {P_2}^{\delta}$, $K := H^{\delta}$ for a random $\delta \in \mathbb{Z}_q$ and $G_1, H \in \mathbb{G}_1$, $P_2 \in \mathbb{G}_2$.

1. One may include vector $\boldsymbol{t} := ({P_1}^{\delta}, {P_1}^{\delta^2}, \ldots, {P_1}^{\delta^k})$ for the users to compute their own witness (to be described). We can have a constant-size public-key (without the vector $\boldsymbol{t}$) when the BA computes the witness for each user using the auxiliary secret value δ.
2. The order of the bilinear groups where $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ should all have the same prime order q, i.e., the accumulator should be instantiated with the same prime number q used in Issuer's parameter.

2.3 Issuing U-Prove Token

The protocol for Issuer to issue U-Prove tokens to Provers is the same as in the original Spec, except that there is a designated attribute x_{id} in the U-Prove token for revocation purpose.

Generating U-Prove token: A U-Prove token has the same form as in Spec:

$$(\mathsf{UID}_{\mathrm{P}},\ h,\ TI,\ PI,\ \sigma'_z,\ \sigma'_c,\ \sigma'_r)$$

where

- The public key h, with a number of attributes $x_t, x_1, \ldots, x_n$ embedded, is $h = (g_0 {g_1}^{x_1} \ldots {g_{\mathsf{id}}}^{x_{\mathsf{id}}} \ldots {g_n}^{x_n} {g_t}^{x_t})^{\alpha} \pmod q$.
- The private key is $\alpha^{-1} \in \mathbb{Z}_q^*$.
- TI/PI are token/prover information field respectively.
- A valid signature from an issuer is given in the form of $(\sigma'_z,\ \sigma'_c,\ \sigma'_r)$.

2.4 Blacklist

A blacklist is published and managed by a BA. Note that BA has the auxiliary information to efficiently compute the accumulator values. For example, on day 1, three elements $x_{\mathsf{id}_1}, x_{\mathsf{id}_2}, x_{\mathsf{id}_3}$ can be accumulated in V_1; then, on day 2, x_{id_2} can be deleted, and x_{id_4} can be added, which results in an updated V_2.

[**For Revocation**] **Maintaining Revocation List:** BA decides an attribute x_{id} for revocation. We can have some options here: x_{id} could uniquely identify a token, a user or an organization. Suppose there are m revocations, BA publishes the list of $\{x_{\mathsf{id}_i}\}$, and accumulates them by

$$V := {P_1}^{\prod_{i=1}^{m} (\delta + x_{\mathsf{id}_i})}.$$

2.5 Presenting U-Prove Tokens

The difference from the basic presentation protocol of a valid U-Prove token is that, now a Prover also needs to prove the knowledge of x_{id}, which is the attribute used for revocation, is not accumulated in BA's blacklist.

Input:

1. Ordered indices of disclosed attributes: $D \subset \{1,\ \ldots,\ n\}$
2. Ordered indices of undisclosed attributes: $U = \{1,\ \ldots,\ n\} \setminus D$
3. U-Prove token: $\mathcal{T} = (\mathsf{UID}_{\mathrm{P}},\ h,\ TI,\ PI,\ \sigma'_z,\ \sigma'_c,\ \sigma'_r)$
4. Messages: $m \in \{0,1\}^*$
5. Private key: $\alpha^{-1} \in \mathbb{Z}_q$
6. Attribute values: $(A_1, \cdots, A_n) \in (\{0,1\}^*)^n$

7. **[For Revocation]** The current accumulator value: V
8. **[For Revocation]** The updated witness: (W, Q, d) where
   - $d = \prod_{i=1}^{m} (\delta + x_{\mathsf{id}_i}) \pmod{(\delta + x_{\mathsf{id}})} \in \mathbb{Z}_q$,
     which can be computed without knowing δ by simple polynomial division of $\prod(z + x_{\mathsf{id}_i})$ by $(z + x_{\mathsf{id}})$, and is non-zero since $x_{\mathsf{id}} \notin \{x_{\mathsf{id}_i}\}$.
   - $W = P_1^{(\prod_{i=1}^{m} (\delta + x_{\mathsf{id}_i}) - d)/(\delta + x_{\mathsf{id}})}$
     which is computable by using vector $\boldsymbol{t}$ in param.
   - $Q = VW^{-x_{\mathsf{id}}} {P_1}^{-d}$.

Proof Generation:

1. **[For Revocation]** Prepare for the commitment part of the proof that the credential is not revoked:

   Randomly pick $x, u, t_1, t_2, t_3, r_x, r_u, r_{t_1}, r_{t_2}, r_{t_3}, r_{\beta_1}, r_{\beta_2}, r_{\beta_3}, r_d, r_{d'} \in \mathbb{Z}_q$. Set:

   $$\begin{array}{llll} X := WH^{t_1}, & Y := QK^{t_1}, & C := {G_1}^{x} H^{u}, A := {G_1}^{r_x} H^{u} \\ R := {G_1}^{t_1} H^{t_2}, & S := {G_1}^{d'} H^{t_3}, & \Gamma := X^{-r_x} H^{r_{\beta_1}} K^{r_{t_1}} {P_1}^{-r_d}, \\ T_1 := {G_1}^{r_{t_1}} H^{r_{t_2}}, & T_2 := {G_1}^{r_{\beta_1}} H^{r_{\beta_2}} R^{-r_x}, & T_3 := {G_1}^{r_{d'}} H^{r_{t_3}}, T_4 := H^{r_{\beta_3}} S^{-r_d}. \end{array}$$

2. **[For Revocation]** (In the original Spec, $a := \mathcal{H}(h^{w_0}(\prod_{i \in U} g_i^{w_i}))$).
   Compute $a := \mathcal{H}(h^{w_0}(\prod_{i \in U} g_i^{w_i}), \mathcal{H}(X, Y, R, S, T_1, T_2, T_3, T_4, \Gamma, \mathsf{param}))$.
3. $c := \mathsf{GenerateChallenge}(IP, \mathcal{T}, a, m, \varnothing, D, \{x_i\}_{i \in D}) \in \mathbb{Z}_q$.
4. **[For Revocation]** Prepare for the response part for the proof that the credential is not revoked, by computing the following:

   $$\begin{array}{llll} \beta_1 := t_1 x_{\mathsf{id}}, & \beta_2 := t_2 x_{\mathsf{id}}, & \beta_3 := t_3 d, & d' := d^{-1} \\ s_u := -cu + r_u, & s_x := -cx + r_x, & s_d := -cd + r_d, & s_{d'} := -cd' + r_{d'} \\ s_{t_1} := -ct_1 + r_{t_1}, & s_{t_2} := -ct_2 + r_{t_2}, & s_{t_3} := -ct_3 + r_{t_2}, & \\ s_{\beta_1} := -c\beta_1 + r_{\beta_1}, & s_{\beta_2} := -c\beta_2 + r_{\beta_2}, & s_{\beta_3} := -c\beta_3 + r_{\beta_3}. & \end{array}$$

5. $x_t := \mathsf{ComputeXt}(IP, TI) \in \mathbb{Z}_q$.
6. For each $i \in \{1, \ldots, n\}$, $x_i := \mathsf{ComputeXi}(IP, A_i) \in \mathbb{Z}_q$.
7. Generate w_0 at random from $\mathbb{Z}_q$, set $r_0 := c\alpha^{-1} + w_0 \pmod{q}$.
8. For each $i \in U$, generate w_i at random from $\mathbb{Z}_q$, set $r_i := -cx_i + w_i \pmod{q}$.
9. Return the U-proven token proof $\left(\{A_i\}_{i \in D},\ a,\ r_0, \{r_i\}_{i \in U}\right)$.
10. **[For Revocation]** Also return the non-revoked proof:

$$(c, s_u, s_x, s_d, s_{d'}, s_{t_1}, s_{t_2}, s_{t_3}, s_{\beta_1}, s_{\beta_2}, s_{\beta_3}, C, X, Y, R, S)\,.$$

Proof Verification:

1. Execute $\mathsf{VerifyTokenSignature}(IP, \mathcal{T})$ which verifies $(\sigma'_z,\ \sigma'_c,\ \sigma'_r)$.
2. $x_t := \mathsf{ComputeXt}(IP,\ TI)$.
3. For each $i \in D$, $x_i := \mathsf{ComputeXi}(IP, A_i)$.
4. Set $c := \mathsf{GenerateChallenge}(IP, \mathcal{T}, a, m, \varnothing, D, \{x_i\}_{i \in D})$.

5. **[For Revocation]** Execute the following steps for non-membership proof verification:

$$\tilde{T_1} := {G_1}^{s_{t_1}} H^{s_{t_2}} R^c,\ \tilde{T_2} := {G_1}^{s_{\beta_1}} H^{s_{\beta_2}} R^{-s_x},$$
$$\tilde{T_3} := {G_1}^{s_{d'}} H^{s_{t_3}} S^c,\ \tilde{T_4} := {G_1}^{-c} H^{s_{\beta_3}} S^{-s_d},$$
$$\tilde{A} := {G_1}^{s_x} H^{s_u} C^c,\quad \tilde{\Gamma} := X^{-s_x} H^{s_{\beta_1}} K^{s_{t_1}} {P_1}^{-s_d} (V^{-1}Y)^c.$$

Verify if $e\,(Y, P_2) \stackrel{?}{=} e(X, P_{pub})$, and if

$$a = \mathcal{H}((g_0 g_t^{x_t} \prod_{i \in D} g_i^{x_i})^{-c} h^{r_0} (\prod_{i \in U} g_i^{r_i}), \mathcal{H}(X, Y, R, S, T_1, T_2, T_3, T_4, \Gamma, \mathsf{param})).$$

3 Crypto-Agile Software Design

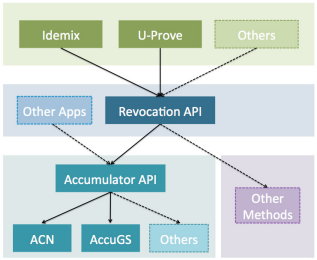

Fig. 1: Our Software Design

The design of our accumulator-and-applications system is illustrated in Figure 1. There is one common application program interface (API) for all accumulators, providing the interface for the operations. Two accumulators, the new one in this paper (based on Fiat-Shamir transformation [13], denoted as ACN) and the existing one (based on Groth-Sahai proof [15], denoted as AccuGS) [1], have been implemented according to the API. They are the first implementation of universal accumulators, and AccuGS provides the only solution for revoking delegatable anonymous credentials not relying on random oracles.

The API could be used to develop accumulator's applications. One such application is for revoking anonymous credentials. We further implement a revocation API. A blacklist authority could use it to create a blacklist and accumulate revoked anonymous credentials, and an user could prove that a credential is not accumulated in a blacklist. Several anonymous credential systems based on prime order could use this revocation. We have used it for U-Prove.

This design supports crypto-agility. With a common accumulator API, it is easy to add other accumulators' implementations based on different assumptions, to replace an existing implementation with a more efficient or safer one, and to switch among them with minimum code refactoring. With a single revocation

API and a single accumulator API, we just need a single implementation of "Revocation using Accumulators". This reduces redundancy and allows us to have a painless changes from, say, AccuGS to ACN.

The system is developed in C++ and built into 3 dynamic link libraries (dll) in Windows. The first, `accumulator.dll`, implements ACN and AccuGS. The second, `RAC.dll`, utilizes the first library for revoking anonymous credentials. The third, `UProveRAC.dll`, uses the second library to integrate revocation into U-Prove. `UProveRAC.dll` consists of all existing UProve API and additional API functions with revocation capability. The new UProveRAC functions allow generating revocation parameters and keys, computing and updating witnesses, and proving and verifying that U-Prove tokens are not revoked.

4 Performance

For a fair comparison, we compare the performances of ACN and the only previous universal accumulator ATSM [3], which derived the membership proof from Nguyen's [19] and introduced a new NM proof. We only compare the proof systems, as other algorithms are very similar. Both the proofs of ACN and ATSM can be made interactive and do not rely on the random oracle heuristics, or can be converted into non-interactive version via Fiat-Shamir heuristics [13] which relies on the random oracle. ATSM only uses symmetric pairing $\mathbb{G}'_1 \times \mathbb{G}'_1 \to \mathbb{G}'_T$, whereas ACN works for both symmetric and asymmetric settings $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. So ACN could use much more efficient asymmetric pairing groups, such as BN curves [14]. Indeed, as discussed, operations in $\mathbb{G}_1$ are much more efficient than corresponding operations in $\mathbb{G}'_1$ and $\mathbb{G}'_T$. ACN is especially suitable for BN curves, which features not only one of the most efficient pairing operations, but also very efficient exponentiation in $\mathbb{G}_1$. Even if we do not perform pre-computation for some pairings, ACN is still significantly more efficient than ATSM. Table 1a compares the numbers of exponentiations and pairings between these proof systems.

Table 1b shows the performance (in milliseconds) of the major functions in ACN and AccuGS [1], running on a modest machine — Intel Core2 2.4 GHz with 4 GB RAM, 64-bit Win7, using 254-bit BN curves. Again, the underlying accumulators are the same so we focus on witness operations. Accumulating 40 elements take 3.5 ms, and updating accumulator with 1 element takes 1.71 ms. Finally, it takes 36.055ms to generate a U-Prove proof and 73.817ms to verify it. Those numbers never depend on the blacklist's size.

References

1. T. Acar and L. Nguyen. Revocation for delegatable anonymous credentials. *PKC 2011 LNCS* 6571, p. 423–440.
2. M. H. Au, W. Susilo, Y. Mu, and S. S. M. Chow. Constant-size dynamic k-times anonymous authentication. *IEEE Systems Journal*, 17(3): 46-57, 2013.
3. M. H. Au, P. P. Tsang, W. Susilo, and Y. Mu. Dynamic universal accumulators for DDH groups and their application to attribute-based anonymous credential systems. *CT-RSA 2009*, *LNCS* 5473, p. 295–308.

Algorithms	ACN		ATSM		
	E_1	$e(\cdot,\cdot)$	E_1'	E_t	$e(\cdot,\cdot)$
NMPrf()	21	0	19	4	2 (+4)
NMVfy()	20	2	19	5	2 (+4)
MemPrf()	14	0	13	3	2 (+3)
MemVfy()	13	2	13	4	2 (+3)

(a) Operation counts: (E_1, E_1', and E_T denote exponentiations in $\mathbb{G}_1$, $\mathbb{G}_1'$, and $\mathbb{G}_T$ resp.; $x(+y)$: x pairings, another y of them can be pre-computed)

Operations	ACN	AccuGS
Update witness	77.102	78.980
NMPrf()	11.882	30.609
NMVfy()	18.714	102.978
MemPrf()	24.148	117.897
MemVfy()	80.635	587.164

(b) Running time ("Update witness" here is for adding 40 and removing 5 elements, i.e., a total of 45 unit updates.)

Table 1: Performance of ACN, ATSM, and AccuGS

4. J. C. Benaloh and M. de Mare. One-way accumulators: A decentralized alternative to digital signatures *EUROCRYPT '93*, *LNCS* 765, p. 274–285.
5. P. Bichsel, J. Camenisch, T. Groß, and V. Shoup. Anonymous credentials on a standard java card. *ACM CCS 2009*, p. 600–610.
6. E. F. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. *ACM CCS 2004*, p. 132–145.
7. J. Camenisch, M. Kohlweiss, and C. Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. *PKC 2009*, *LNCS* 5443, p. 481–500.
8. J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. *CRYPTO 2002*, *LNCS* 2442, p. 61-76.
9. J. Camenisch and E. Van Herreweghen. Design and implementation of the idemix anonymous credential system. *ACM CCS 2002*, p. 21–30.
10. S. S. M. Chow, J. K. Liu, and D. S. Wong. Robust receipt-free election system with ballot secrecy and verifiability. *NDSS 2008*, p. 81–94.
11. S. S. M. Chow, W. Susilo, and T. H. Yuen. Escrowed linkability of ring signatures and its applications. *VIETCRYPT 2006*, *LNCS* 4341, p. 175–192.
12. Y. Dodis, A. Kiayias, A. Nicolosi, and V. Shoup. Anonymous identification in ad hoc groups. *EUROCRYPT 2004*, *LNCS* 3027, p. 609–626.
13. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. *CRYPTO '86*, *LNCS* 263, p. 186–194.
14. C. C. F. P. Geovandro, M. A. S. Jr., M. Naehrig, and P. S. L. M. Barreto. A family of implementation-friendly BN elliptic curves. *Journal of Systems and Software*, 84(8):1319–1326, 2011.
15. J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. *EUROCRYPT 2008*, *LNCS* 4965, p. 415–432.
16. J. Li, N. Li, and R. Xue. Universal accumulators with efficient nonmembership proofs. *ACNS 2007*, *LNCS* 4521, p. 253–269.
17. Microsoft. U-Prove cryptographic specification V1.1 Revision 2. `http://research.microsoft.com/en-us/projects/u-prove`, 2013. (Last Visited on May 6$^{\text{th}}$, 2013.)
18. D. Nelson. Crypto-agility requirements for remote authentication dial-in user service (RADIUS). RFC 6421 (Informational), Nov. 2011.
19. L. Nguyen. Accumulators from bilinear pairings and applications. *CT-RSA 2005*, *LNCS* 3376, p. 275–292.