# Anonymous Authentication with Revocation and Encryption Scheme for V2X

Taha Adeel Mohammed

December 23, 2022

## 1 Introduction

Below we describe a scheme that allows registered users to authenticate themselves with an RSU, and once succesfully authenticated, derive a symmetric key with the help of the RSU and communicate with its adjacent peers using CAMs encrypted using that key.

We also describe a way in which the RSU/IA can revoke the authentication of a user in a privacy preserving manner if it suspects that the user is malicious.

## 2 Previous Works

This schemes builds on the Anonymous Authentication and Encryption Scheme proposed by Kanchan[3], and adds anonymous revocation to it by integrating and adapting the PEREA[5] scheme into it.

## 3 High Level Overview

The user has secret $\alpha, \beta$ in $\mathbb{Z}_q^*$, which it uses to register with the IA, getting credentials $(a, b, c, d)$, which can be used for authentication and deriving the symmetric key for CAMs during authentication.

Additionally, to support revocation of misbehaving users, we have the user also present a fresh ticket $t_k$ to the RSU for each authentication, and maintain a queue of the past $K$ tickets it used for authentication. And we have the RSU maintain a blacklist of tickets belonging to misbehaving users.

During authentication, we have the user prove in zero-knowledge that the last $K$ tickets in its queue haven't been blacklisted by the IA. We also have the user prove in zero knowledge that the queue of last $K$ tickets is correctly formed. After verifying the user's credentials, the integrity of the queue, and that the user hasn't been blacklisted, the RSU computes a witness for the ticket $t_k$ and a signature on the queue and shares it with the user. The user can then use this signature and witness during its next authentication to prove to the RSU in zero-knowledge of the validity of his credentials.

## 4 Building Blocks

### 4.1 Preliminaries

#### Notation

- $a \overset{R}{\leftarrow} X$ : Denotes that a is chosen uniformly at random from the set X
- $\mathbb{Z}_q$ : Set of integers modulo q
- $\Lambda_l$ : Set of integers of size atmost l-bits.
- $\Pi_l$ : Set of primes of size atmost l-bits.
- $\Delta(l, \delta)$ : The set of integers $\{2^{l-1}, \ldots, 2^{l-1} + 2^{\delta} - 1\}$
- *Safe Prime*: $p$ such that $p$ and $\frac{p-1}{2}$ are both prime.
- *l-bit safe prime product*: Product of two $\lfloor \frac{l}{2} \rfloor$-bit safe primes.
- $QR_N$ : Set of quadratic residues modulo $N$.
- $\phi(N)$ : Euler's totient of $N$.

#### Hardness Assumptions

The security of the accumulator, signature scheme, and ZKPoKs used for the revocation depends on the below hardness assumptions. Let $N$ be a random $\lambda$-bit safe prime product.

- The **Strong RSA Assumption** says that there exists no PPT algorithm which, on input $N$ and $u \in \mathbb{Z}_N^*$ , returns $e > 1$ and $v$ such that $v^e = u \pmod N$, with non-negligible probability (in $\lambda$).

- The **Decisional Diffie-Hellman(DDH) Assumption** says that there exists no PPT algorithm which, on input quadruple $(g, g^a, g^b, g^c) \in QR_N^4$,, where $a, b \overset{R}{\leftarrow} \mathbb{Z}_{|QR_N|}$, and $c \overset{R}{\leftarrow} Z_{|QR_N|}$ or $c = ab$ with equal probability, correctly distinguishes which is the case with probability non-negligibly (in $\lambda$) greater than 1/2.

**ZKPoK Protocols**

A *Zero-Knowledge Proof-of-Knowledge (ZKPoK)* protocol is a protocol in which a prover convinces a verifier that some statement is true without the verifier learning anything except the truth of the statement.

The notation notation introduced by Camenisch and Stradler[2] is used to describe the ZKPoK protocols used in our scheme. For example, $PK\{(x) : y = g^x\}$ denotes a ZKPoK protocol that proves knowledge of an integer $x$ such that $y = g^x$ holds, without revealing $x$.

## 4.2 Tickets and Queues

The user picks a ticket uniformly at random from the set $\Pi_{l_t}$, where $l_t = 166$. i.e $t \overset{R}{\leftarrow} \Pi_{l_t}$. As $|\Pi_{l_t}| > 2^{160}$, the probability of two randomly chosen tickets being equal is atmost $2^{-80}$, which is negligible. The ticket domain is set as $\mathcal{T} = \{-2^{l_T} + 1, \ldots, 2^{l_T} - 1\}$, where $l_T = 330$. Setting $\mathcal{T} \supsetneq \Pi_{l_t}$ allows the users who know a ticket in $\Pi_{l_t}$ to efficiently prove in zero knowledge that they know some ticket in $\mathcal{T}$.

The user maintains a queue $Q$ of the last $K$ tickets it used for authentication, where $K$ is the revocation window size. The queue supports enqueueing (`Enq`) and dequing (`Deq`) operations. We have $Q[i]$ denote the $i^{th}$ least ticket recently enqueued ticket in the queue $Q$. The domain of $Q$ is $\mathcal{Q} = \mathcal{T}^{K+1}$.

## 4.3 Accumulator Scheme for Tickets

We require that the users should be able to successfully prove that they have not been blacklisted by the IA, without revealing their tickets. Hence for this, we make use of Universal Dynamic Accumulators (UDAs) introduced by Li et al.[4], which allows for an efficient zero-knowledge proof of non-membership, in time independent of number of accumulated values.

Below we describe a construction of UDAs adapted for our requirements and notation. We call the modified scheme `TicketAcc`.

### Key Generation

- Input security parameter: $\texttt{param}_{acc} = l_N$ ( $= 1024$ recommended).
- Pick $N = pq$, an $l_N$-bit safe prime product, and $g \overset{R}{\leftarrow} QR_N$
- Output the accumulator's public key $pk_{acc} = (l_N, N, g)$ and secret key $sk_{acc} = \phi(N)$.

### Accumulating Tickets

- For ticket $t \in \mathcal{T}$ and accumalator value $V$, we have

$$Accumulate(V, t) \rightarrow V' = V^t \pmod N \tag{1}$$

- Similarly, for set $S_T = \{t_1, t_2, \ldots, t_L\}$ and accumulator value $V$, we have

$$Accumulate(V, S_T) \rightarrow V' = V^{t_1 \cdot t_2 \cdot \cdots \cdot t_L} \pmod N \tag{2}$$

- The accumalator value is initallyg. Hence we define

$$Accumulate(S_T) = Accumulate(g, S_T) = g^{t_1 \cdot t_2 \cdot \cdots \cdot t_L} \pmod N \tag{3}$$

### Non-membership Witnesses

- If $V = Accumulate(S_T)$ for some $S_T \subset \mathcal{T}$ and $t \in \mathcal{T} \setminus S_t$, then there exists a non-membership witness $w = (a, d) \in \mathbb{Z}_{\lfloor N/4 \rfloor} \times QR_N$ for $t$ with respect to $V$ such that $1 = IsNonMember(t, V, w)$ holds, where

$$IsNonMember(t, V, w) = \begin{cases} 1 & \text{if } V^a \equiv d^t g \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

- The computation of such a witness can be done using knowledge of $sk_{acc}$ as

$$ComputeWitness(t, V, sk_{acc}) \to w \tag{5}$$

Refer to [4] for the implementation details and unforgeability proof of above function.

- A valid prover can convince a verifier that $t$ was not accumulated in $V$ using $w$, without revealing $t$ or $w$, by conducting

$$PK\{(t, w) : 1 = IsNonMember(t, V, w)\} \tag{6}$$

The construction of above protocol and its security proof have been given by Li et al.[4]. The construction of above ZKPoK has a complexity of $O(1)$, i.e. independent of the number of accumulated values.

**Update of Non-membership Witnesses**

- Given witness $w$ for ticket $t$ such that $1 = IsNonMember(t, V, w)$ holds, when the accumulator value $V$ is updated via the accumulation of a new ticket $t' \in \mathcal{T} \setminus \{t\}$ into it, one can compute, without knowledge of $sk_{acc}$, an updated witness $w'$ for $t$ such that $IsNonMember(t, V', w') = 1$ holds, as

$$UpdateWitness(w, t, V, t') \to w' \tag{7}$$

Again, refer to [4] for the implementation details of above function.

- Similarly, for a set $S_T \subset \mathcal{T} \setminus \{t\}$, we define

$$UpdateWitness(w, t, V, S_T) \to w' \tag{8}$$

to denote the repetitive invocation of $UpdateWitness$ to update $w$ for $t$ when tickets in $S_T$ are accumulated into $V$, one at a time, in any order.

- Hence the complexity of $UpdateWitness$ is $O(|S_T|)$, i.e linear in the number of new values added to the accumulator.

## 4.4 Protocol for Queue Signing

The RSU needs to verify the integrity of the queue, since otherwise a user can circumvent revocation by fabricating a queue with an incorrect set of K tickets. For this, the RSU signs the queue on a succesful authentication so that it can be convinced of its integrity during the next authentication.

This must be done without revealing the queue or the signature, as otherwise the user's actions can be linked. Hence for this, we use the signature scheme given by Camenisch and Lysyanskaya[1] for blocks of messages, adapted for our notation, given by `QueueSig` below.

### Key Generation

- Input security parameter: $\texttt{param}_{sig} = (l_N, l_s, l_e, l_T, l, \delta_r)$.
- Pick $N = pq$, an $l_N$-bit safe prime product, and $b, c, g_0, \ldots, g_K \overset{R}{\leftarrow} QR_N$
- Output the signature schemes's public key $pk_{sig} = (param_{sig}, N, b, c, (g_i)_{i=0}^{K})$ and secret key $sk_{acc} = \phi(N)$.

### Request for Signature

To request a signature on a commited queue $Q = (t_i)_{i=0}^{K} \in \mathcal{Q}$, the user picks $r \overset{R}{\leftarrow} \Delta(l_N, \delta_r)$ and commits Q as:

$$Commit(Q, r) \rightarrow C = c^r \prod_{i=0}^{K} g_i^{t_i} \pmod N \tag{9}$$

and sends the commitment $C$ to the RSU.

***Proof of Correctness:*** The user then proves to the RSU that $C$ is a valid commitment on $Q$ by conducting the below protocol. The RSU proceeds only if the protocol is successful.

$$PK\{(Q, r) : C = Commit(Q, r) \ \wedge \ Q \in \mathcal{Q} \ \wedge \ r \in \Delta(l_N, \delta_r)\} \tag{10}$$

This protocol can be constructed using standard protocols for proving relations among components of a discrete logarithm representation of a group of elements.

**Signing**

The RSU signs and returns to the user a signature $\tilde{\sigma}$ on $C$ usign its private key $sk_{sig}$ as:

$$Sign(sk_{sig}, C) \rightarrow \tilde{\sigma} = (r', e, v), \tag{11}$$

where $r' \overset{R}{\leftarrow} \Lambda_{l_s},\ e \overset{R}{\leftarrow} \Lambda_{l_e},$ and $v = (bc^{r'}C)^{1/e \pmod{\phi(N)}} \pmod N$

**Finalizing**

The user finalizes the signature $\tilde{\sigma} = (r', e, v)$ on the commitment $C$ into a signature $\sigma$ on the queue $Q$ as:

$$Finalize(\tilde{\sigma}, r) \rightarrow \sigma = (r + r', e, v) \tag{12}$$

The user proceeds only if the signature verifies, i.e $Verify(Q, \sigma) = 1$ holds, where

$$Verify(Q, \sigma) = \begin{cases} 1 & \text{if } v^e \equiv bc^s \prod_{i=0}^{K} g_i^{t_i} \ \wedge\ e > 2^{l_e - 1} \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

### 4.4.1 Security and Complexity Analysis

The construction of above protocol has an $O(K)$ computational and communication complexity between the user and RSU. Additionally, we require that the signatures are unforgeable and that the RSU can't deduce the tickets in the queue or the users identity during the protocol. When $l_N \geq 1024, l \geq 160, l_e > l_T + 2, l_s = l_N + l_T + l$ and $\delta_r = \lfloor \frac{l_N - 1}{\epsilon} - l \rfloor$ for some $1 < \epsilon \in \mathbb{R}$, the security properties of the scheme hold under the strong RSA assumption, as shown in the original CL signature scheme on blocks of messages. [1].

### 4.4.2 Proof of Knowledge of a signed queue

The below protocol allows a user to prove to the RSU the possession of a valid signature without revealing the queue and signature themselves.

$$PK\{(Q, \sigma) : 1 = Verify(Q, \sigma)\ \wedge\ Q \in \mathcal{Q}\} \tag{14}$$

The original CL signature scheme on blocks of messages [1] can be referred to for the construction and security proof of above protocol.

### 4.4.3 Proof of relation between two queues

During authentication, the user updates his queue from $Q'$ to $Q = Q'.Enq(t*).Deq()$ for use during the next authentication, where $t^*$ is the new ticket. The below protocol allows the user to prove to the RSU that the new queue $Q$ is a valid update of the old queue $Q'$.

$$PK\{(Q', Q, t^*) : Q = Q'.Enq(t*).Deq() \ \wedge \ Q' \in \mathcal{Q} \ \wedge \ t^* \in \mathcal{T}\} \tag{15}$$

This protocol can be constructed as follows. The user picks $r_0, r_1 \overset{R}{\leftarrow} \Delta(l_N, \delta_r)$ and commits both $Q'$ and $Q$ using eq. 9 and conduct the below protocol with the RSU, which can be done similarly to protocol 10.

$$PK\{(r_0, r_1, (t_i)_{i=0}^{K+1}) : \ \wedge_{b=0,1} \ C_b \equiv c^{r_b} \prod_{i=0}^{K} g_i^{t_{i+b}}\} \tag{16}$$

# 5 Construction

Below we describe the construction of the Anonymous Authentication with Revocation and Encryption Scheme using the building blocks described in the previous section.

## 5.1 Setup

For the randomizable signatures using bilinear pairings, the IA, given security parameter $k$, outputs the Type-3 bilinear pairing parameters $(e, g_1, g_2, g, G_1, G_2, G) \overset{R}{\leftarrow} Gen(1^k)$. The IA then generates $sk_{bp} = (x, y) \overset{R}{\leftarrow} \mathbb{Z}_q^*$ and $pk_{bp} = (X = g_2^x, Y = g_2^y)$.

The IA decides on an appropriate revocation window size $K$, and on input parameters $param_{acc}$ and $param_{sig}$, generates $(sk_{acc}, pkacc)$ and $(sk_{sig}, pk_{sig})$ respectively, according to the schemes in previous section. The IA also picks a prime $\hat{t} \overset{R}{\leftarrow} \Pi_{l_t}$ to be used as the default value to fill the user's queue during registration. Then the IA initializes its blacklist as $BL = \phi$, accumalator value as $V = g$ and ticket-list as $TL = \phi$.

Finally, the IA creates its private key as $sk_{IA} = (sk_{bp}, sk_{acc}, sk_{sig})$ and publishes its public key as $pk_{IA} = (pk_{bp}, pk_{acc}, pk_{sig}, K, \hat{t})$.

## 5.2 Registration

The user has secrets $(\alpha, \beta) \xleftarrow{R} \mathbb{Z}_q^*$ and sends $req = (a = g_1^{\beta}, b = a^{\alpha})$ to the issuer. The issuer verifies the vehicle and uses $sk_{bp}$ to compute $(c, d) = (a^x, (b \cdot c)^y)$ and outputs the signature $\sigma_{bp} = (a, b, c, d)$ to the user.

Additionally, to support revocation

- `(Request for credential)`: The user picks $t^* \xleftarrow{R} \Pi_{l_t}$ and initializes their queue $Q'$ as $Q' = \{\hat{t}, \hat{t}, \ldots, \hat{t}, t\}$. Then the user sends $C = Commit(Q', r)$, where $r \xleftarrow{R} \Delta(l_N, \delta_r)$, to the IA.

- `(Proof of correctness)`: The user then engages in the below protocol with the IA to prove that $Q'$ was initialized correctly, and that the commitment $C$ was correctly computed. (Construction can be done similarly to protocol 10)

$$PK\{(Q', r) : \wedge_{i=0}^{K-1} \hat{t} = Q'[i] \ \wedge \ C = Commit(Q', r) \ \wedge \ Q' \in \mathcal{Q}\} \quad (17)$$

- `(Credential issuing)`: The IA computes $\tilde{\sigma}_q = Sign(C, sk_{acc})$ and $\hat{w} = ComputeWitness(\hat{t}, V', sk_{acc})$, where $V' = Accumulate(BL')$. The IA returns $(\tilde{\sigma}_q, \hat{w}, BL', V')$ to the user.

- `(Credential finalizing)`: The user computes $\sigma_q = Finalize(\tilde{\sigma}_q, r)$ and proceeds only if $V' = Accumulate(BL')$, $1 = Verify(Q', \sigma_q)$ and $1 = IsNonMember(\hat{t}, V', \hat{w})$ hold.

The user then stores its credentials as

$$cred = (\sigma_{bp}, Q', \sigma_q, (w'_i = \hat{w})_{i=0}^{K-1}, BL', V') \quad (18)$$

## 5.3 Authentication

We now describe the authentication protocol between the user and the RSU.

***Randamizable Signature Check:***
The user shares $\sigma_{bp}^r = (a^r, b^r, c^r, d^r)$, which the RSU verifies by checking that $e(a^r, X) = e(c^r, g_2)$ and $e(d^r, g_2) = e(bc^r, Y)$. Then the RSU can optionally engage in the protocol for key generation and encryption(as decribed in [3]), with a unique symmetric key for this authentication, so that the further steps in the authentication protocol happen over a secure channel.

***Blacklist examination***
The user obtains the current blacklist BL from the RSU, and after verifying that none of the ticekts in $Q$ have been blacklisted, the user finds $\Delta_{BL} = BL \setminus BL'$, i.e set of newly blacklisted tickets.

***Request for authentication***
The user generates a new ticket $t^* \overset{R}{\leftarrow} \Pi_{l_t}$ and $r \overset{R}{\leftarrow} \Delta(l_N, \delta_r)$, and computes:

$$t_K = Q'[K] \tag{19}$$
$$Q = Q'.Enq(t^*).Deq() \tag{20}$$
$$C = Commit(Q, r) \tag{21}$$
$$V = Accumulate(V', \Delta_{BL}) \tag{22}$$
$$w_i = WitnessUpdate(w_i', Q'[i], V', \Delta_{BL}) \ \forall \ i \in [0, K-1] \tag{23}$$

The user then sends $(t_K, C)$ to the RSU, and after the RSU verifies that $t_K$ is a fresh prime in $\Pi_{l_t}$, it adds $t_K$ to the ticket-list $TL$.

`Proof of correctness:` The user then engages in the below ZKPoK with the RSU to convince the RSU that he hasn't been revoked yet, $t_K$ is a well formed ticket, and that $C$ is a well formed commitment on the user's next queue.

$$\begin{aligned}PK\{(Q', \sigma_q', (w_i)_{i=0}^{K-1}, t^*, Q, r) : \\ t_K &= Q'[K] && \wedge \\ 1 &= Verify(Q', \sigma_q') && \wedge \\ \wedge_{i=0}^{K-1} \ 1 &= IsNonMember(Q'[i], V, w_i) && \wedge \\ Q &= Q'.Enq(t^*).Deq() && \wedge \\ C &= Commit(Q', r) && \wedge \\ Q &\in \mathcal{Q} \ \wedge \ t^* \in \mathcal{T} && \}\end{aligned} \tag{24}$$

Above protocol can be built using the individual protocols from the previous sections.

***Refreshment issuing***
The RSU then computes and shares the following with the user:

$$\tilde{\sigma_q} = Sign(t^*, sk_{acc}) \tag{25}$$
$$w^* = ComputeWitness(V, t^*, sk_{acc}) \tag{26}$$

***Credential refreshment***
The user finalizes the signature $\sigma_q$ as

$$\sigma_q = Finalize(\tilde{\sigma}_q, Q, r) \tag{27}$$

and checks for correctness by verifying that $1 = Verify(Q, \sigma_q)$ and $1 = IsNonMember(t^*, V, w^*)$ hold. Then the user updates its credentials as $cred = (\sigma_{bp}, (Q', \sigma_q), (w'_i)_{i=0}^{K-1}, BL', V')$, where:

$$\begin{aligned}(Q', \sigma'_q) &\leftarrow (Q, \sigma_q)\\ (w'_0, w'_1, \ldots, w'_{K-2}) &\leftarrow (w'_1, w'_2, \ldots, w'_{K-1})\\ w'_{K-1} &\leftarrow w^*\\ (BL', V') &\leftarrow (BL, V)\end{aligned}$$

## 5.4 Revocation

To blacklist a misbehaving user who provided $t_k$, the RSU/IA updates the blacklist as $BL' = BL \cup \{t_k\}$, and updates the accumulator as $V' = Accumulate(V, t_k)$. The next time the misbehaving user tries to authenticate, he will be unable to prove that $t_k$ from his queue is not in the accumulator. Hence he won't be authenticated.

## 5.5 Key generation and Encryption Scheme

After the authentication in the above steps, the rest of the construction for key generation and sharing of encrypted CAMs can be done similarly to the original scheme [3].

## 6 Conclusion

We have shown that the original scheme[3] can be extended to support revocation for malicious users while still preserving the privacy of the users. However, the currently proposed scheme significantly impacts the efficiency of the scheme during the authentication phase, since for each authentication, the user now has to perform a ZKPoK for each value in his queue, siginificantly increasing the computation and communication overheads. These overheads might not be acceptable for V2X applications.

## References

[1] J. Camenisch and A. Lysyanskaya. "A Signature Scheme with Efficient Protocols." In: *SCN* 2576 (2002).

[2] J. Camenisch and M. Stadler. "Efficient group signature schemes for large groups". In: *CRYPTO* 1294 (1997).

[3] Kanchan. "Lightweight Privacy Preserving Authentication and Encryption Scheme for V2X". In: (2022).

[4] J. Li, N. Li, and R. Xue. "Universal accumulators with efficient nonmembership proofs". In: (2007).

[5] Patrick P. Tsang, Man Ho Au Apu, and Kapadia Sean W. Smith. "PEREA: Towards Practical TTP-Free Revocation in Anonymous Authentication". In: (2009).