Predicates in the Unicity Infrastructure

Ahto Buldas

October 2025

1 Introduction

Predicates generalize the concepts of token ownership and transfer in the Unicity infrastructure, which in the paper [1] were defined via digital signatures as follows:

- *Owner* – a (legal/physical) person who controls the private key of a digital signature scheme
- *Ownership condition* – the public key pk that corresponds to the private key of the owner.
- *Transfer* – the owner presents a digital signature σ on $m = H(h_{\mathsf{st}}, h_{\mathsf{tx}})$ (where $h_{\mathsf{st}}, h_{\mathsf{tx}}$ are the state hash and the transaction hash, respectively) such that $\mathsf{V}(\mathsf{pk}, m, \sigma) = 1$.

In this paper, we present the following generalized concepts:

- *Owner* – an abstract group of (legal/physical) persons that together control the information necessary to create the next transaction with the token. The information may include private keys.
- *Ownership condition* – a logical condition (predicate) ν.
- *Transfer* – the owner (as a group) presents a bit-string σ such that the condition $\nu(\tau, m, \sigma) = 1$ holds, where $m = H(h_{\mathsf{st}}, h_{\mathsf{tx}})$ and τ is the *system time* (an integer defined by the Unicity service). This means that predicates may also put restrictions on transaction execution time.

So far, we only have used predicates of type $\nu(\tau, m, \sigma) \equiv \mathsf{V}(\mathsf{pk}, m, \sigma)$, i.e. all predicates ν are in the form $\mathsf{V}(\mathsf{pk}, \cdot, \cdot)$.

The predicates approach is certainly not new and is used already in the Bitcoin blockchain, where ν is called the *locking script* and σ is called the *unlocking script* or *witness*.

A typical example of a generalized predicate is the *delayed execution* predicate defined by $\nu(\tau, m, \sigma) \equiv \mathsf{dex}_{\mathsf{pk},\tau_0}(\tau, m, \sigma) \equiv (\tau \geq \tau_0) \ \wedge \ \mathsf{V}(\mathsf{pk}, m, \sigma) = 1$, which states that the next transaction can be executed by the owner of the private key of pk not earlier than τ_0. The delayed execution predicate is used in the protocols for inter-blockchain *atomic swaps* between Bitcoin type blockchains.

It is a natural question whether the security properties (no double spending, no blocking, no association) will still hold in the Unicity infrastructure if the generalized predicates are in use. We will show in this paper that double-spending is indeed impossible in the generalized scheme. However, the non-blocking condition is much less obvious.

The main concern is that arbitrarily chosen predicates do not have guaranteed security properties like the UF-CMA condition for digital signatures. For example, if a user chooses the ownership predicate $\nu(\tau, m, \sigma) \equiv (\sigma^2 - 2\sigma + 1 = 0)$ then anyone who is able to solve quadratic equations can make the next transfer with the token and hence, the no-blocking condition may seem to be violated. On the other hand, by intentionally choosing such an ownership condition, the previous owner may indicate that the next abstract owner of the token is the group of all people who can efficiently solve quadratic equations, and in this sense, intuitively, the no blocking condition is not violated.

The predicates can also be chosen so that they cannot be solved (satisfied) in principle, i.e. they are logically inconsistent. For example $\nu(\tau, m, \sigma) \equiv (\sigma^2 + 2\sigma + 1 = 0)$, where σ is required to be real number, cannot be satisfied.

Another (less trivial) example of improper use of predicates is when a user applies one-time signature scheme as a many-times signature scheme. In this case, security-critical information leaks gradually so that an adversary, having triples $(\tau_1, m_1, \sigma_1), \ldots, (\tau_n, m_n, \sigma_n)$ (so that $\nu(\tau_i, m_i, \sigma_i) = 1$), can construct a new triple (τ, m, σ) such that $\nu(\tau, m, \sigma) = 1$ and $m \notin \{m_1, \ldots, m_n\}$.

Therefore, the precise mathematical formulation of the no-blocking condition – *only the owner of the private key of* pk *can block the state* $S = (\mathsf{pk}, h_{\mathsf{st}})$ – must be revisited. We will redefine the no-blocking condition as follows – *only those who can solve the predicate* ν *can block the state* $S = (\nu, h_{\mathsf{st}})$, where by solving ν we mean finding, for a given m, a pair (τ, σ) so that $\nu(\tau, m, \sigma) = 1$. In this paper, we will make this security definition precise and prove that it holds in the Unicity infrastructure. Intuitively, this means that undesired blocking can happen only because of the weakness of the user-chosen predicates and never because of the structural weakness of the Unicity infrastructure itself.

2 Unicity Infrastructure with Predicates

2.1 Unicity Service

Like in the paper [1] we model the Unicity Service as an ideal functionality, where we use *system time* as an additional state variable. The *Unicity Service* US is modeled as a state machine with state (R, τ), where:

- R is a key-value store (dictionary), where both keys and values are of type $\{0,1\}^{|k|}$. Initially, $R = \emptyset$. We will write $R[k] = \perp$ if there are no pairs (k, v) stored in R.
- τ is a non-negative integer called the *system time*. Initially, $\tau = 0$.

There are two types of inputs of US:

- A request Q is a tuple $(\nu, h_{\mathsf{st}}, h_{\mathsf{tx}}, \sigma)$, where:
  - ν is a predicate that represents the current abstract owner of a token, i.e. the owner before the transaction with the hash h_{tx} is executed;
  - h_{st} of type $\{0,1\}^k$ is a state hash: the hash of the state of the token before the transaction is executed;
  - h_{tx} of type $\{0,1\}^k$ is a transaction data hash (defined later);
  - σ of type $\{0,1\}^s$ is a digital signature of the transaction.
- A *new time* input message NT that contains a non-negative integer τ_{new}.

The request $Q = (\nu, h_{\mathsf{st}}, h_{\mathsf{tx}}, \sigma)$ is processed by US with the state (R, τ) as follows:

1. If $R[H(\nu, h_{\mathsf{st}})] = \perp$ and $\nu(\tau, H(h_{\mathsf{st}}, h_{\mathsf{tx}}), \sigma) = 1$ then

$$R \leftarrow R \cup \{(H(\nu, h_{\mathsf{st}}), h_{\mathsf{tx}})\} \ ,$$

   i.e. the new value of R is defined by setting $R[H(\nu, h_{\mathsf{st}})] \leftarrow h_{\mathsf{tx}}$ and leaving the rest of the contents of R unchanged.

2. A proof π_{inc} of the statement $R[H(\nu, h_{\mathsf{st}})] = v$ (inclusion proof) is returned.

During the normal work of the system, NT is initiated by (and can only executed by) the consensus layer of the system. In attack scenarios, we will also give the adversary the access to the NT functionality. This is for making the security conditions stronger.

The new time input $\mathsf{NT} = (\tau_{\mathsf{new}})$ is processed by US with the state (R, τ) as follows: If $\tau_{\mathsf{new}} > \tau$ then $\tau \leftarrow \tau_{\mathsf{new}}$, i.e. the system time is updated if the new value is larger than the current value.

It is easy to see that if $(R_0, \tau_0) = (\emptyset, 0)$ is the initial state, $I_1, I_2, \ldots, I_n$ is any sequence of inputs (requests or new time inputs), R_i is the dictionary after the input I_i, and τ_i is the system time after the input I_i, then:

- $\tau_0 \leq \tau_1 \leq \ldots \leq \tau_n$, i.e. the system time never decreases.
- $R_0 \subseteq R_1 \subseteq \ldots \subseteq R_n$, i.e. the elements are never removed.
- The state R_n is a *partial function*, i.e. $\{(k, v), (k, v')\} \subseteq R$ implies $v = v'$.

We say that a key k is *blocked* if $R[k] \neq \perp$.

2.2 Verification Function

We assume that the inclusion proofs π_{inc} contain the system time τ, i.e. the value of τ when the corresponding new element was set in R. There is an extraction function time that extracts the system time from the proof π_{inc}, i.e. $\tau \leftarrow \mathsf{time}(\pi_{\mathsf{inc}})$. We assume that the inclusion proofs π_{inc} can be verified by any party using a verification function $\mathcal{V}$ so that:

- If $\mathcal{V}(k, v; \pi_{\mathsf{inc}}) = 1$ then $R[k] = v$ in the current state (R, τ) of US. Hence, as R is a partial function, for every $k, v, v', \pi_{\mathsf{inc}}, \pi'_{\mathsf{inc}}$, the following implication holds:

$$\mathcal{V}(k, v; \pi_{\mathsf{inc}}) = \mathcal{V}(k, v'; \pi'_{\mathsf{inc}}) = 1 \quad \Rightarrow \quad v = v' \ . \tag{1}$$

- If $R[H(\nu, h_{\mathsf{st}})] = h_{\mathsf{tx}}$ after a request $\pi_{\mathsf{inc}} \leftarrow \mathsf{US}(\nu, h_{\mathsf{st}}, h_{\mathsf{tx}}, \sigma)$ to the Unicity Service, then $\mathcal{V}(H(\nu, h_{\mathsf{st}}), h_{\mathsf{tx}}, \pi_{\mathsf{inc}}) = 1$.
- If a request $\pi_{\mathsf{inc}} \leftarrow \mathsf{US}(Q)$ was processed in the state $S = (R, \tau)$ that changes $R[H(\nu, h_{\mathsf{st}})]$ from $\perp$ to $\neq \perp$, then $\mathsf{time}(\pi_{\mathsf{inc}}) = \tau$.

2.3 Transactions with a Token

Every token has an abstract owner A represented by a predicate ν. We will call the pair (ν, aux) the *state* of the token and $h_{\mathsf{st}} = H(\nu, \mathsf{aux})$ the state hash of the token.

The transaction payload (before certifying the transaction) with the token is a pair $T = (h_{\mathsf{st}}, D)$, where:

1. h_{st} is the state hash before executing the transaction,
2. D (transaction data) contains the following fields:
   - ν': the predicate of the next abstract owner,
   - x: a uniformly chosen random string $x \leftarrow \{0,1\}^{\ell}$,
   - aux': other data for the next state.

The pair (ν', aux') defines the next state of the token after executing the transaction T. The next state hash is $h'_{\mathsf{st}} = H(h_{\mathsf{st}}, x)$.

Certifying a transaction $T = (h_{\mathsf{st}}, D)$ involves the following steps:

1. $(h_{\mathsf{tx}}, d) \leftarrow \mathsf{Com}(H(D))$ is computed using a perfectly hiding commitment scheme $(\mathsf{Set}, \mathsf{Com}, \mathsf{Open})$. The commitment h_{tx} is called the *transaction data hash*.
2. The hash value $h_T = H(h_{\mathsf{st}}, h_{\mathsf{tx}})$ is computed.
3. A solution σ is created such that $\nu(\tau_{\mathsf{exp}}, h_T, \sigma) = 1$ for an expected[1] value τ_{exp} of system time.
4. The query $Q = (\nu, h_{\mathsf{st}}, h_{\mathsf{tx}}, \sigma)$ is created.
5. US is called to obtain $\pi \leftarrow \mathsf{US}(Q)$.
6. The certified transaction $(T, \sigma, h_{\mathsf{tx}}, d, \pi)$ is formed.

[1] In order to avoid failed certification calls, users could (1) query the current time from US to minimize the difference, and (2) prioritize safety in predicate design. For example, the delayed execution predicate $\mathsf{dex}_{\mathsf{pk},\tau_0}$ from the introduction uses an inequality $\tau \geq \tau_0$, which remains satisfiable for all future times.

Verifying a certified transaction A certified transaction $(T, \sigma, h_{\mathsf{tx}}, d, \pi)$ is verified in the state (ν, h) by the following algorithm:

$\mathcal{V}_{\mathsf{cert}}(T, \sigma, h_{\mathsf{tx}}, d, \pi; \nu, h)$: If at least one of the following checks fail, return 0, otherwise return 1:

1. $T.h_{\mathsf{st}} = h$;
2. $\mathsf{Open}(h_{\mathsf{tx}}, d) = H(T.D)$;
3. $\nu(\mathsf{time}(\pi), H(h_{\mathsf{st}}, h_{\mathsf{tx}}), \sigma) = 1$;
4. $\mathcal{V}(H(\nu, T.h_{\mathsf{st}}), h_{\mathsf{tx}}, \pi) = 1$.

Definition 2.1 (certification in a state) *A tuple* $(T, \sigma, h_{\mathsf{tx}}, d, \pi)$ *is said to be* certified in state (ν, h) *iff* $\mathcal{V}_{\mathsf{cert}}(T, \sigma, h_{\mathsf{tx}}, d, \pi; \nu, h) = 1$.

Mint transactions are the same as in the paper ["Unicity Execution Layer"], i.e. they do not use generalized predicates.

2.4 Token Ledger

A *token ledger* is a sequence

$$(T_0, \sigma_0, \pi_0; h^1_{\mathsf{st}}), (T_1, \sigma_1, h^1_{\mathsf{tx}}, d_1, \pi_1; h^2_{\mathsf{st}}), \ldots, (T_n, \sigma_n, h^n_{\mathsf{tx}}, d_n, \pi_n; h^{n+1}_{\mathsf{st}})$$

where (T_0, σ_0, π_0) is a certified mint transaction and for every index $i = 1, \ldots, n$:

1. $(T_i, \sigma_i, h^i_{\mathsf{tx}}, d_i, \pi_i)$ is a certified transaction in the state $(T_{i-1}.D.\nu', h^i_{\mathsf{st}})$;
2. $h^i_{\mathsf{st}} = H(h^{i-1}_{\mathsf{st}}, x_{i-1})$ where $x_{i-1} = T_{i-1}.D.x$.

3 Security

Consider a token with the state $S = (\nu, \mathsf{aux})$. The transfer protocol ensures the following properties:

- *No double-spending*: Only one certified transaction can be created in the state S.
- *No association*: the Unicity service is unable to identify the transactions with the same token.
- *No blocking*: Only the owner of the private key of ν can block the state $S = (\nu, \mathsf{aux})$ if it was not blocked before.

In this section, we present security proofs for all three properties. The proofs of no double-spending and no association are very similar to the proofs in [1]. The no blocking property had to be modified to cover arbitrary predicates.

3.1 Security against Double-Spending

A double-spending adversary uses US as an oracle.

Double-spending scenario involves the following steps:

1. $(T, \sigma, h_{\mathsf{tx}}, d, \pi_{\mathsf{inc}}), (T', \sigma', h'_{\mathsf{tx}}, d', \pi'_{\mathsf{inc}}), (\nu, h) \leftarrow A^{\mathsf{US}}$.
2. The attack is successful iff $T \neq T'$ and

$$\mathcal{V}_{\mathsf{cert}}(T, \sigma, h_{\mathsf{tx}}, d, \pi_{\mathsf{inc}}; \nu, h) = \mathcal{V}_{\mathsf{cert}}(T', \sigma', h'_{\mathsf{tx}}, d', \pi'_{\mathsf{inc}}; \nu, h) = 1 \ . \quad (2)$$

Definition 3.1 (Double-spending security) *The Unicity Service is said to be S-secure against double-spending if it has S as a security profile in the double-spending scenario.*

Analysis: If the adversary is successful, then from (2) and the definition of $\mathcal{V}_{\mathsf{cert}}$ it follows that $T.h_{\mathsf{st}} = T'.h_{\mathsf{st}} = h$ and:

$$\mathcal{V}(H(\nu, h), h_{\mathsf{tx}}; \pi_{\mathsf{inc}}) = \mathcal{V}(H(\nu, h), h'_{\mathsf{tx}}; \pi'_{\mathsf{inc}}) = 1 \ ,$$

which implies $h_{\mathsf{tx}} = h'_{\mathsf{tx}}$ by equation (1). From Def. 2.1 it also follows that $\mathsf{Open}(h_{\mathsf{tx}}, d) = H(T.D)$ and $\mathsf{Open}(h_{\mathsf{tx}}, d') = \mathsf{Open}(h'_{\mathsf{tx}}, d') = H(T'.D)$. From $(h, T.D) = (T.h_{\mathsf{st}}, T.D) = T \neq T' = (T'.h_{\mathsf{st}}, T'.D) = (h, T'.D)$ it follows that $T.D \neq T'.D$. Hence, we have two cases:

a) $H(T.D) = H(T'.D)$, which means that a collision has been found for H.
b) $H(T.D) \neq H(T'.D)$, which implies $\mathsf{Open}(h_{\mathsf{tx}}, d) = H(T.D) \neq H(T'.D) = \mathsf{Open}(h_{\mathsf{tx}}, d')$ and hence, the commitment h_{tx} has been opened in two different ways.

Theorem 3.1 *If H is S-secure collision-resistant and the commitment scheme is S-secure computationally binding, then the Unicity service is S_{double}-secure against double-spending, where $S_{\mathsf{double}}(\epsilon) = \frac{S(\epsilon/2)}{t_{\mathsf{ver}}+1}$ and t_{ver} is the predicate verification time.*

Proof. Let A be a t-time double-spending adversary that succeeds with probability ϵ. We construct a collision-finder A_{coll} for the hash function and a double-opening adversary A_{com} for the commitment scheme as follows:

- A_{coll} proceeds as follows:
  1. Simulate $(T, \sigma, h_{\mathsf{tx}}, d, \pi_{\mathsf{inc}}), (T', \sigma', h'_{\mathsf{tx}}, d', \pi'_{\mathsf{inc}}), (\nu, h) \leftarrow A^{\mathsf{US}}$ by maintaining its own version of US.
  2. Output the pair $(T.D, T'.D)$.

  The computational overhead function of A_{coll} is $\tau_{\mathsf{coll}}(t) = (t_{\mathsf{ver}} + 1) \cdot t$, because simulating a US query requires one signature verification and the number of calls is limited by the running time t of A.
- A_{com} proceeds as follows:

1. Simulate $(T, \sigma, h_{\mathsf{tx}}, d, \pi_{\mathsf{inc}}), (T', \sigma', h'_{\mathsf{tx}}, d', \pi'_{\mathsf{inc}}), (\nu, h) \leftarrow A^{\mathsf{US}}$ by maintaining its own version of US.
2. Output the triple (h_{tx}, d, d').

The computational overhead function of A_{com} is the same as that of A_{coll}, i.e. $\tau_{\mathsf{com}}(t) = \tau_{\mathsf{coll}}(t) = (t_{\mathsf{ver}} + 1) \cdot t$.

If A succeeds, then in case a) the collision finder A_{coll} succeeds, and in case b) the double-opener A_{com} succeeds. Hence, $\epsilon \leq \epsilon_{\mathsf{coll}} + \epsilon_{\mathsf{com}}$, where ϵ_{coll} is the success probability of A_{coll} and ϵ_{com} is the success probability of A_{com}. Therefore, the function S_{double} defined by

$$S_{\mathsf{double}}(\epsilon) = \tau_{\mathsf{coll}}^{-1}(S(\epsilon/2)) = \frac{S(\epsilon/2)}{t_{\mathsf{ver}} + 1}$$

is a security profile of the Unicity service against double-spending. □

3.2 Security against Association

The Association adversary $A = (A_1, A_2)$ is two-stage.

Association scenario involves the following steps:

1. $(h_{\mathsf{st}}, \nu', \mathsf{aux}', a) \leftarrow A_1$.
2. $x \leftarrow \{0,1\}^{\ell}$.
3. $h'_{\mathsf{st}} \leftarrow H(h_{\mathsf{st}}, x)$.
4. $h_{\mathsf{tx}} \leftarrow \mathsf{Com}^{c}(H(\nu', x, \mathsf{aux}')))$.
5. $x' \leftarrow A_2(a; h'_{\mathsf{st}}, h_{\mathsf{tx}})$.
6. The attack is successful iff $h_{\mathsf{st}} \in \{0,1\}^{k}$, $x' \in \{0,1\}^{\ell}$, and $H(h_{\mathsf{st}}, x') = h'_{\mathsf{st}}$. The success ϵ of A is the probability that the attack is successful.

Definition 3.2 (association security) *The Unicity Service is said to be S-secure against association if it has S as a security profile in the association scenario.*

Theorem 3.2 *If the hash function is S-secure (k, ℓ)-one-way and the commitment scheme is perfectly hiding, then the Unicity Service is S_{assoc}-secure against association, where $S_{\mathsf{assoc}}(\epsilon) = S(\epsilon) - t_{\mathsf{sm}} - t_{\mathsf{hash}} - t_{\mathsf{com}}$, where t_{sm}, t_{hash}, t_{com} are the random sampling time, the hashing time, and the commitment computation time, respectively.*

Proof. Let $A = (A_1, A_2)$ be a t-time adversary that succeeds in the association scenario with probability ϵ. Consider the following modified attack scenario:

1. $(h_{\mathsf{st}}, \nu', \mathsf{aux}', a) \leftarrow A_1$.

2. $x \leftarrow \{0,1\}^{\ell}$.
3. $h'_{\mathsf{st}} \leftarrow H(h_{\mathsf{st}}, x)$.
4. $x'' \leftarrow \{0,1\}^{\ell}$.
5. $h'_{\mathsf{tx}} \leftarrow \mathsf{Com}^{c}(H(\nu', x'', \mathsf{aux}')))$.
6. $x' \leftarrow A_2(a; h'_{\mathsf{st}}, h'_{\mathsf{tx}})$.
7. The attack is successful iff $h_{\mathsf{st}} \in \{0,1\}^{k}$, $x' \in \{0,1\}^{\ell}$, and $H(h_{\mathsf{st}}, x') = h'_{\mathsf{st}}$.

For any fixed value of $L = (h_{\mathsf{st}}, \nu', \mathsf{aux}', a)$, due to perfect hiding, commitments $h_{\mathsf{tx}} = \mathsf{Com}^{c}(H(\nu', x, \mathsf{aux}'))$ and $h'_{\mathsf{tx}} = \mathsf{Com}^{c}(H(\nu', x'', \mathsf{aux}'))$ have equal probability distributions. Moreover, by Lemma **??** (with $g(x) = H(\nu', x, \mathsf{aux}')$), the random variables x and $h_{\mathsf{tx}} = \mathsf{Com}^{c}(H(\nu', x, \mathsf{aux}'))$ are independent.

Since x'' and x are independent, the commitment $h'_{\mathsf{tx}} = \mathsf{Com}^{c}(H(\nu', x'', \mathsf{aux}'))$ is independent of both x and $h'_{\mathsf{st}} = H(h_{\mathsf{st}}, x)$. Therefore, the joint distributions of $(h'_{\mathsf{st}}, h_{\mathsf{tx}})$ and $(h'_{\mathsf{st}}, h'_{\mathsf{tx}})$ are equal, and hence A succeeds in the modified scenario with probability ϵ. We construct an adversary $A' = (A'_1, A'_2)$ as follows:

- A'_1 proceeds as follows:
  1. $(h_{\mathsf{st}}, \nu', \mathsf{aux}', a) \leftarrow A_1$;
  2. return (h_{st}, a'), where $a' = (\nu', \mathsf{aux}', a)$.
- $A'_2(a'; y)$ with $a' = (\nu', \mathsf{aux}', a)$ proceeds as follows:
  1. $x'' \leftarrow \{0,1\}^{\ell}$;
  2. $h'_{\mathsf{tx}} \leftarrow \mathsf{Com}^{c}(H(\nu', x'', \mathsf{aux}')))$;
  3. $x' \leftarrow A_2(a; y, h'_{\mathsf{tx}})$;
  4. return x'.

The computational time overhead function of A' is $\tau(t) = t + t_{\mathsf{sm}} + t_{\mathsf{hash}} + t_{\mathsf{com}}$ and hence, the function S_{assoc} defined by

$$S_{\mathsf{assoc}}(\epsilon) = \tau^{-1}(S(\epsilon)) = S(\epsilon) - t_{\mathsf{sm}} - t_{\mathsf{hash}} - t_{\mathsf{com}}$$

is a security profile of the Unicity service against association. □

3.3 Security against Blocking

By a predicate family we mean a pair $(\mathsf{G}_{\mathsf{pr}}, \mathsf{S}_{\mathsf{pr}})$ of algorithms so that:

- $(\mathsf{sk}, \nu) \leftarrow \mathsf{G}_{\mathsf{pr}}$ generates a private key sk and a predicate ν.
- $\mathsf{S}_{\mathsf{pr}}(\mathsf{sk}, m)$ solves the predicate for m, i.e. either $\perp \leftarrow \mathsf{S}_{\mathsf{pr}}(\mathsf{sk}, m)$ (the solver gives up) or $(\tau, \sigma) \leftarrow \mathsf{S}_{\mathsf{pr}}(\mathsf{sk}, m)$ such that $\nu(\tau, m, \sigma) = 1$.

The case $\perp \leftarrow \mathsf{S_{pr}}(\mathsf{sk}, m)$ is necessary because the predicates can potentially be chosen so that they cannot be satisfied.

A predicate solving adversary $A^{\mathcal{O}}_{\mathsf{solve}}$ for the predicate family $(\mathsf{G_{pr}}, \mathsf{S_{pr}})$ uses an oracle $\mathcal{O}$ that uses (initially empty) dictionaries $\mathsf{sk}[\iota]$, $\nu[\iota]$ as the state and answers to two types of queries:

- $\mathcal{O}(\mathsf{gen}; \iota)$ – a generation query that returns $\perp$ if $\nu[\iota] \neq \perp$, and otherwise generates $(\mathsf{sk}, \nu) \leftarrow \mathsf{G_{pr}}$, sets $\mathsf{sk}[\iota] \leftarrow \mathsf{sk}$, $\nu[\iota] \leftarrow \nu$, and returns ν.
- $\mathcal{O}(\mathsf{solve}; \iota, m)$ – a solving query that returns $\perp$ if either $\nu[\iota] = \perp$ or $\mathsf{S_{pr}}(\mathsf{sk}[\iota], m) = \perp$, and otherwise, if $(\tau, \sigma) \leftarrow \mathsf{S_{pr}}(\mathsf{sk}[\iota], m)$, it returns (τ, σ).

The predicate solving scenario involves the following steps:

1. $(\iota, \tau, m, \sigma) \leftarrow A^{\mathcal{O}}_{\mathsf{solve}}$
2. The attack is successful if:
   a) $\nu[\iota] \neq \perp$, i.e. the query $\mathcal{O}(\mathsf{gen}; \iota)$ was made by $A^{\mathcal{O}}_{\mathsf{solve}}$.
   b) $\nu[\iota](\tau, m, \sigma) = 1$
   c) All queries of the form $\mathcal{O}(\mathsf{solve}; \iota, m)$ made by $A^{\mathcal{O}}_{\mathsf{solve}}$ (if there were any) were answered with $\perp$.

A blocking adversary A uses two oracles:

1. US: the Unicity Service,
2. TS: that uses (initially empty) dictionaries $\mathsf{sk}[\iota]$, $\nu[\iota]$ as the state and answers to two types of queries:
   - $\mathsf{TS}(\mathsf{gen}; \iota)$ – a generation query that returns $\perp$ if $\mathsf{sk}[\iota] \neq \perp$, and otherwise generates $(\mathsf{sk}, \nu) \leftarrow \mathsf{G_{pr}}$, sets $\mathsf{sk}[\iota] \leftarrow \mathsf{sk}$, $\nu[\iota] \leftarrow \nu$, and returns ν.
   - $\mathsf{TS}(\mathsf{solve}; \iota, h, D)$ – a solving query that returns $\perp$ if either $\mathsf{sk}[\iota] = \perp$ or $\mathsf{S_{pr}}(\mathsf{sk}[\iota], H(h, h_{\mathsf{tx}})) = \perp$, and otherwise, returns $(\tau, \sigma, h_{\mathsf{tx}}, d)$, where $(h_{\mathsf{tx}}, d) \leftarrow \mathsf{Com}(H(D))$ and $(\tau, \sigma) \leftarrow \mathsf{S_{pr}}(\mathsf{sk}[\iota], H(h, h_{\mathsf{tx}}))$.

Blocking scenario involves the following steps:

1. $(\iota, h_{\mathsf{st}}) \leftarrow A^{\mathsf{US},\mathsf{TS}}$
2. A is successful if:
   a) $\nu[\iota] \neq \perp$
   b) $R[H(\nu[\iota], h_{\mathsf{st}})] \neq \perp$ after the scenario
   c) No (successful) queries of the form $\mathsf{TS}(\mathsf{solve}; \iota, h_{\mathsf{st}}, D)$ were made.

   The success ϵ of A is the probability that the attack is successful

Note that if such a query was made, then the request $Q = (\nu[\iota], h_{\mathsf{st}}, h_{\mathsf{tx}}, \sigma)$ to US at system time τ will trivially ensure $R[H(\nu[\iota], h_{\mathsf{st}})] \neq \perp$. Note that the adversary can set the system time appropriately before the request Q. Hence, this is excluded by the security condition.

Definition 3.3 (blocking security) *The Unicity Service is said to be S-secure against blocking if it has S as a security profile in the blocking scenario.*

Analysis: The adversary A can be successful in the following cases:

a) A request $Q = (\nu', h'_{\mathsf{st}}, h_{\mathsf{tx}}, \sigma)$ with $(\nu', h'_{\mathsf{st}}) \neq (\nu[\iota], h_{\mathsf{st}})$ to US enforces $R[H(\nu[\iota], h_{\mathsf{st}})] \neq \perp$, which means that $H(\nu[\iota], h_{\mathsf{st}}) = H(\nu', h'_{\mathsf{st}})$ and hence, a collision for H was found.

b) A request $Q = (\nu[\iota], h_{\mathsf{st}}, h_{\mathsf{tx}}, \sigma)$ to US enforces $R[H(\nu[\iota], h_{\mathsf{st}})] \neq \perp$, which implies $\nu[\iota](\tau, H(h_{\mathsf{st}}, h_{\mathsf{tx}}), \sigma) = 1$ from the description of US. Then we have two possibilities:

   b1) A query $(\tau', \sigma', h'_{\mathsf{tx}}, d) \leftarrow \mathsf{TS}(\mathsf{solve}; \iota, h'_{\mathsf{st}}, D)$ was made such that the equality $H(h'_{\mathsf{st}}, h'_{\mathsf{tx}}) = H(h_{\mathsf{st}}, h_{\mathsf{tx}})$ holds. From the success condition of A it follows that $h'_{\mathsf{st}} \neq h_{\mathsf{st}}$ and we have a collision for H.

   b2) If no queries $(\tau', \sigma', h'_{\mathsf{tx}}, d) \leftarrow \mathsf{TS}(\mathsf{solve}; \iota, h'_{\mathsf{st}}, D)$ were made with $H(h'_{\mathsf{st}}, h'_{\mathsf{tx}}) = H(h_{\mathsf{st}}, h_{\mathsf{tx}})$ then this means that A was able to solve the predicate family, i.e. for $m = H(h_{\mathsf{st}}, h_{\mathsf{tx}})$ finds τ, σ so that $\nu[\iota](\tau, m, \sigma) = 1$ without using the predicate solving functionality.

Theorem 3.3 *If the predicate family $(\mathsf{G}_{\mathsf{pr}}, \mathsf{S}_{\mathsf{pr}})$ is S-secure against solving and the hash function is S-secure collision-resistant, then the Unicity service is S_{block}-secure against blocking, where $S_{\mathsf{block}}(\epsilon) = \frac{S(\epsilon/2)}{\max\{t_{\mathsf{gen}}, t_{\mathsf{sig}}, t_{\mathsf{ver}}, t_{\mathsf{ver}}, t_{\mathsf{com}}, t_{\mathsf{hash}}\}}$ and t_{gen}, t_{sig}, $t_{\mathsf{ver}}, t_{\mathsf{com}}, t_{\mathsf{hash}}$ are the key generation time (for G_{pr}), solving time (for S_{pr}), verification time (for ν), commitment time, and hashing time, respectively.*

Proof. Let A be a t-time blocking adversary that succeeds with probability ϵ. We construct a collision-finder A_{coll} and a solver $A^{\mathcal{O}}_{\mathsf{solve}}$ as follows:

- A_{coll} proceeds as follows:
  1. Simulates $(\iota, h_{\mathsf{st}}) \leftarrow A^{\mathsf{US},\mathsf{TS}}$ and records all the oracle queries.
  2. If $A^{\mathsf{US},\mathsf{TS}}$ was successful and either the case a) or b1) occurs, A_{coll} outputs the collision that is guaranteed in this case.

  The oracles are simulated as follows:
  - US-queries: A_{coll} maintains its own version of R.
  - TS-queries: A_{coll} directly uses G_{pr} and S_{pr}.

  The computational time overhead function for the construction of A_{coll} is $\tau_{\mathsf{coll}}(t) = \max\{t_{\mathsf{gen}}, t_{\mathsf{sig}}, t_{\mathsf{ver}}\} \cdot t$, where t_{ver} is the predicate verification time (for US-queries), t_{sig} is the predicate solving time (for $\mathsf{TS}(\mathsf{solve}; \cdot)$-queries), and t_{gen} is the generation time (for $\mathsf{TS}(\mathsf{gen}; \cdot)$-queries).
- $A^{\mathcal{O}}_{\mathsf{solve}}$ proceeds as follows:
  1. Simulates $(\iota, h_{\mathsf{st}}) \leftarrow A^{\mathsf{US},\mathsf{TS}}$ and records all the oracle queries.
  2. If $A^{\mathsf{US},\mathsf{TS}}$ was successful and b2) occurs and $Q = (\nu', h_{\mathsf{st}}, h_{\mathsf{tx}}, \sigma)$ was the request that enforces $R[H(\nu[\iota], h_{\mathsf{st}})] \neq \perp$ at system time τ then:
     3. $m \leftarrow H(h_{\mathsf{st}}, h_{\mathsf{tx}})$.

4. Output (ι, τ, m, σ).

The oracles are simulated as follows:

- US-queries are simulated so that $A^{\mathcal{O}}_{\mathsf{solve}}$ maintains its own version of R.
- US-queries are simulated so that $A^{\mathcal{O}}_{\mathsf{solve}}$ maintains its own copy of the dictionary $\nu[]$ and processes the queries as follows:
  - $\mathsf{TS}(\mathsf{gen}; \iota)$ – If $\nu[\iota] \neq \perp$ then return $\perp$. Otherwise, call $\nu[\iota] \leftarrow \mathcal{O}(\mathsf{gen}; \iota)$ and return $\nu[\iota]$.
  - $\mathsf{TS}(\mathsf{solve}; \iota, h, D)$ – If $\nu[\iota] = \perp$ then return $\perp$. Otherwise, compute $(h_{\mathsf{tx}}, d) \leftarrow \mathsf{Com}(H(D))$, call $v \leftarrow \mathcal{O}(\mathsf{solve}; \iota, H(h, h_{\mathsf{tx}}))$ and return v. Note that either $v = \perp$ or $v = (\tau, \sigma)$.

As the request Q was accepted by US and it changes $R[H(\nu[\iota], h_{\mathsf{st}})]$, we have $\nu' = \nu[\iota]$ (description of US) and $\nu[\iota](\tau, m, \sigma) = 1$. Note that in the case b2) the request $\mathsf{TS}(\mathsf{solve}; \iota, h_{\mathsf{st}}, D)$ with $\mathsf{Open}(h_{\mathsf{tx}}, d) = H(D)$ was never made which also means that the query $\mathcal{O}(\mathsf{solve}; \iota, m)$ with $m = H(h_{\mathsf{st}}, h_{\mathsf{tx}})$ was never made, and hence, $A^{\mathcal{O}}_{\mathsf{solve}}$ is successful in the predicate solving scenario. The computational time overhead function for the construction of $A^{\mathcal{O}}_{\mathsf{solve}}$ is

$$\tau_{\mathsf{solve}}(t) = \max\{t_{\mathsf{ver}}, t_{\mathsf{com}}\} \cdot t + t_{\mathsf{hash}} \leq \max\{t_{\mathsf{ver}}, t_{\mathsf{com}}, t_{\mathsf{hash}}\} \cdot t$$

if $t \geq 1$, where t_{ver} is the signature verification time (for US queries), t_{com} is the commitment time (for $\mathsf{TS}(\mathsf{solve}; \cdot)$-queries) and t_{hash} is the hash computation time (for output).

If A succeeds, then either A_{coll} or $A^{\mathcal{O}}_{\mathsf{solve}}$ succeeds and hence $\epsilon \leq \epsilon_{\mathsf{coll}} + \epsilon_{\mathsf{solve}}$. As for the function $\tau(t) = \max\{t_{\mathsf{gen}}, t_{\mathsf{sig}}, t_{\mathsf{ver}}, t_{\mathsf{ver}}, t_{\mathsf{com}}, t_{\mathsf{hash}}\} \cdot t$ the inequalities $\tau_{\mathsf{coll}}(t) \leq \tau(t)$ and $\tau_{\mathsf{solve}}(t) \leq \tau(t)$ hold, we imply that

$$S_{\mathsf{block}}(\epsilon) = \tau^{-1}(S(\epsilon/2)) = \frac{S(\epsilon/2)}{\max\{t_{\mathsf{gen}}, t_{\mathsf{sig}}, t_{\mathsf{ver}}, t_{\mathsf{ver}}, t_{\mathsf{com}}, t_{\mathsf{hash}}\}}$$

is a security profile of the Unicity Service against blocking. □

References

[1] Buldas, A., Truu, A., Laanoja, R., Rogojin, V.: The Unicity Execution Layer. A manuscript (2025)