Security Analysis of the NuID Decentralized Identification Protocol

Jonathan Katz

1 Background

We provide an analysis of a decentralized identification protocol used by NuID. Relevant background is provided here, but we refer to the document describing the protocol for further details and discussion.[1]

1.1 Requirements and Definitions

Recall that an *identification protocol* consists of algorithms $(\mathsf{KeyGen}, \mathcal{P}, \mathcal{V})$ as follows:[2]

- The *key-generation algorithm* KeyGen takes as input a user's secret information sk and outputs a corresponding public value pk.
- The *proof-generation algorithm* $\mathcal{P}$ takes as input the secret information sk and a challenge (or "nonce") nonce, and outputs a proof π.
- The *proof-verification algorithm* $\mathcal{V}$ takes as input public information pk, a nonce nonce, and a proof π, and outputs either 0 or 1 (indicating rejection or acceptance, respectively).

(There may also be public parameters, left implicit in the above, that can be used by any of the algorithms.) It is required that for all secrets sk, all public values pk output by $\mathsf{KeyGen}(\mathsf{sk})$, any nonce nonce, and any proof π output by $\mathcal{P}(\mathsf{sk}, \mathsf{nonce})$ it holds that $\mathcal{V}(\mathsf{pk}, \mathsf{nonce}, \pi) = 1$.

An identification protocol as above would be used in the following way:

1. A user runs $\mathsf{KeyGen}(\mathsf{sk})$ using their secret information sk in order to generate the public value pk. The user then appends pk to a ledger and receives a unique, persistent identifier ID derived from the ledger transaction.
2. When the user wishes to identify itself to some service, the user and service carry out the following steps:
   (a) The user sends their claimed identity ID to the service.
   (b) The service searches the ledger for the appropriate transaction $(\mathsf{ID}, \mathsf{pk})$. It then generates a nonce nonce and sends it to the user.
   (c) The user computes $\pi \leftarrow \mathcal{P}(\mathsf{sk}, \mathsf{nonce})$ and sends π to the service.
   (d) The service accepts the user's claimed identity ID iff $\mathcal{V}(\mathsf{pk}, \mathsf{nonce}, \pi) = 1$.

[1] NuID: A Model for Trustless, Decentralized Authentication and Self-Sovereign Identity. 2020. https://nuid.io/pdf/white-paper.pdf.

[2] Notation has been changed from the NuID White Paper, and the interfaces to the algorithms have been simplified.

1.2 Threat Model

The goal of the protocol is to prevent an attacker from being able to impersonate an honest user. We consider an attacker who may do any and all of the following:

- Observe public transactions posted to the ledger.
- Impersonate a service to a user.
- Eavesdrop on executions of the identification protocol between a user and a service. (In fact, it is easy to see that eavesdropping is subsumed by being able to impersonate a service to a user, as long as the mechanism by which a service generates a nonce is public.)
- Attempt to impersonate a user to a service.

Informally speaking, we require that even after doing the above it should remain infeasible for an attacker to falsely impersonate an honest user. We formalize this via a definition that gives the attacker access to various *oracles* modeling its ability to carry out the above attacks.

Definition 1 *Fix some identification protocol* $\Pi = (\mathsf{KeyGen}, \mathcal{P}, \mathcal{V})$ *and consider the following experiment involving* Π *and an attacker* $\mathcal{A}$*:*

1. *A secret value* sk *is chosen from some distribution* $\mathcal{D}$*.*
2. *Compute* $\mathsf{pk} \leftarrow \mathsf{KeyGen}(\mathsf{sk})$ *and give* pk *to* $\mathcal{A}$*.*
3. $\mathcal{A}$ *may repeatedly query a* proof oracle $\mathcal{P}(\mathsf{sk}, \cdot)$ *that allows it to specify an arbitrary nonce* nonce *and receive in return a proof computed using* sk *and* nonce*.*
4. *At some point,* $\mathcal{A}$ *requests a challenge nonce. In response, a nonce* nonce^* *is generated and given to* $\mathcal{A}$*, who in turn outputs a proof* π*.*

$\mathcal{A}$ succeeds *if* $\mathcal{V}(\mathsf{pk}, \mathsf{nonce}^*, \pi) = 1$*. Let* $\mathsf{Adv}^1_{\Pi,\mathcal{D}}(\mathcal{A})$ *denote the probability that* $\mathcal{A}$ *succeeds.*

For a given identification protocol we would like to bound the probability with which an attacker $\mathcal{A}$ running for some specified amount of time succeeds. Note, however, that this is not possible unless two additional elements are specified: (1) the probability distribution $\mathcal{D}$ from which sk is chosen (if it is easy for the attacker to predict sk, then the attacker can trivially generate a valid proof π); and (2) how nonce^* is generated (if it is generated in a predictable way, then the attacker can succeed by querying nonce^* to its proof oracle in step 3 of the attack). We will address the first point explicitly in our later analysis. Regarding the second point, we assume that services generate nonces from uniform κ-bit strings.

1.3 Candidate Protocol

Let H denote a cryptographic hash function, and let $\mathbb{G}$ be a cyclic group of prime order q with generator g. The protocol Π under consideration from the previous document is defined as follows:

- $\mathsf{KeyGen}(\mathsf{sk})$ computes $x := H(\mathsf{sk})$ and outputs $\mathsf{pk} := g^x$.
- $\mathcal{P}(\mathsf{sk}, \mathsf{nonce})$ first computes $x := H(\mathsf{sk})$. It then chooses uniform $r \in \mathbb{Z}_q$ and sets $A := g^r$. Finally, it computes $c := H(g^x, \mathsf{nonce}, A)$ followed by $s := c \cdot x + r \bmod q$. It outputs the proof (c, s).

- $\mathcal{V}(\mathsf{pk}, \mathsf{nonce}, \pi)$, where $\pi = (c, s)$, works as follows: compute $A := g^s/\mathsf{pk}^c$; then output 1 if and only if $H(\mathsf{pk}, \mathsf{nonce}, A) \stackrel{?}{=} c$.

2 Security Analysis

2.1 Zero Knowledge

We first prove that the protocol Π from Section 1.3 is *zero knowledge* when H is modeled as a random oracle. This means that an execution of the protocol leaks nothing about sk beyond what is already revealed by pk. This, in turn, implies that attacker's ability to interact with the proof oracle does not help the attacker determine sk or impersonate the user to a service. The fact that the protocol is zero knowledge greatly simplifies the security proof of the overall protocol.

To prove that the protocol is zero knowledge, we show how a *simulator* who is given the ability to program the hash function H (as is the case when H is modeled as a random oracle) can simulate proofs without knowing sk. The simulator—who is given pk and nonce but not sk—works as follows:

1. Choose uniform $c, s \in \mathbb{Z}_q$.
2. Set $A := g^s/\mathsf{pk}^c$.
3. Program $H(\mathsf{pk}, \mathsf{nonce}, A)$ to be equal to c. (If $H(\mathsf{pk}, \mathsf{nonce}, A)$ is already defined, then this step causes a simulation failure. We show below that this occurs with negligible probability.)
4. Output the proof (c, s).

Let $\mathsf{Collision}$ denote the event that A was used in a previous hash query by the attacker or in a previously simulated proof. Since A is uniform, the probability that $\mathsf{Collision}$ occurs, above, is at most $(q_H + q_{\mathcal{P}})/q$, where q_H is the number of H-queries made by the attacker and $q_{\mathcal{P}}$ is the number of previously simulated proofs. By a union bound, the probability that $\mathsf{Collision}$ ever occurs is thus at most $q_{\mathcal{P}} \cdot (q_H + q_{\mathcal{P}})/q$.

As we now show, if $\mathsf{Collision}$ does not occur then a proof output by the simulator is distributed *identically* to a proof generated by the real prover (who knows sk). To see this, note that when the real prover generates a proof (c, s) and $\mathsf{Collision}$ does not occur, we have:

- the auxiliary value A is a uniform element of $\mathbb{G}$;
- $H(\mathsf{pk}, \mathsf{nonce}, A) = c$, where c is a uniform element of $\mathbb{Z}_q$;
- s is uniquely determined as $s = c \cdot x + \log_g A$.

On the other hand, for proofs generated by the simulator we have:

- c is a uniform element of $\mathbb{Z}_q$ and $H(\mathsf{pk}, \mathsf{nonce}, A) = c$;
- A is a uniform element of $\mathbb{G}$ (because s is uniform);
- s is uniquely determined as $s = c \cdot x + \log_g A$.

Clearly, the distributions are identical.

The fact that the protocol is zero knowledge means that in considering Definition 1 we may ignore step 3. (We make this more formal in Section 2.3.)

2.2 Proof of Knowledge

We also claim that the protocol is a *proof of knowledge* when H is modeled as a random oracle. Roughly speaking, this means that if an attacker is able to generate a valid proof relative to pk, then the attacker must know $\log_g \mathsf{pk}$.

The basic idea is as follows. (This intuition can be turned into a formal proof, as we do in Section 2.3, using standard techniques.) Consider an adversary who queries $H(\mathsf{pk}, \mathsf{nonce}, A)$, receives in response a uniform value c, and then outputs a proof $\pi = (c, s)$ for which $A = g^s/\mathsf{pk}^c$. We then *rewind* the adversary and return a second, uniform value c' in response to its query $H(\mathsf{pk}, \mathsf{nonce}, A)$. If $c' \neq c$ and the attacker outputs a second proof $\pi' = (c', s')$ for which $A = g^{s'}/\mathsf{pk}^{c'}$, then we can compute $\log_g \mathsf{pk}$ as follows. Since $g^s/\mathsf{pk}^c = A = g^{s'}/\mathsf{pk}^{c'}$, we have

$$g^{s-s'} = \mathsf{pk}^{c-c'}.$$

It follows that $\log_g \mathsf{pk} = (s - s')/(c - c') \bmod q$.

2.3 Reducing Security to Guessing the Secret

Here we consider a security definition that encapsulates the core of the identification protocol.

Definition 2 *Consider the following experiment involving an attacker $\mathcal{A}$:*

1. *A secret value sk is chosen from some distribution $\mathcal{D}$.*
2. *Compute $x := H(\mathsf{sk})$ and $\mathsf{pk} := g^x$. Give pk to $\mathcal{A}$.*
3. *$\mathcal{A}$ outputs a value x'.*

$\mathcal{A}$ $\mathsf{succeeds}$ if $x' = x$. Let $\mathsf{Adv}^2_{\mathcal{D}}(\mathcal{A})$ be the probability that $\mathcal{A}$ succeeds.

Note that $\mathcal{A}$ succeeds if it guesses x; it is not required to guess sk. This is because knowledge of x is sufficient to impersonate the user. It is worth informally observing, however, that there are two ways $\mathcal{A}$ can determine x: either by guessing sk (in which case it can compute $x = H(\mathsf{sk})$ and then verify that x is correct using pk), or by directly computing $\log_g \mathsf{pk}$.

The results of the previous two sections allow us to relate the success probability of an adversary attacking identification protocol Π (i.e., with respect to Definition 1) to the success probability of an adversary guessing x (cf. Definition 2).

Theorem 1. *Fix a distribution $\mathcal{D}$ and an adversary $\mathcal{A}_1$ attacking Π in the sense of Definition 1, where H is modeled as a random oracle and $\mathcal{A}_1$ makes $q_{\mathcal{P}}$ proof queries and q_H hash queries. Then there is an adversary $\mathcal{A}_2$ in the sense of Definition 2 whose expected running time is roughly[3] the same as that of $\mathcal{A}_1$, and for which $\mathsf{Adv}^2_{\mathcal{D}}(\mathcal{A}_2) \geq \mathsf{Adv}^1_{\Pi,\mathcal{D}}(\mathcal{A}_1) - q_{\mathcal{P}} \cdot (q_H + q_{\mathcal{P}})/q - q_{\mathcal{P}}/2^{\kappa}$.*

Proof. Fix some adversary $\mathcal{A}_1$ attacking identification protocol Π (in the sense of Definition 1), and making q_H queries to H and $q_{\mathcal{P}}$ queries to the proof oracle. Define $\epsilon_1 = \mathsf{Adv}^1_{\Pi,\mathcal{D}}(\mathcal{A}_1)$.

Let Succ denote the event that $\mathcal{A}_1$ outputs a valid proof *and* nonce^* is not one of the nonces used in $\mathcal{A}_1$'s queries to the proof oracle; define $\epsilon_2 = \Pr[\mathsf{Succ}]$. We have $\epsilon_2 \geq \epsilon_1 - q_{\mathcal{P}}/2^{\kappa}$ (assuming nonces are chosen as described earlier).

[3]The exact dependence of the running time of $\mathcal{A}_2$ on the running time of $\mathcal{A}_1$ can be deduced from the proof.

Next consider replacing the proof oracle with the zero-knowledge simulator from Section 2.1, and let ϵ_3 denote the probability of Succ in this modified experiment. As long as a simulation failure does not occur the simulation is perfect, and therefore $\epsilon_3 \geq \epsilon_2 - q_{\mathcal{P}} \cdot (q_H + q_{\mathcal{P}})/q$.

Now construct the following adversary $\mathcal{A}_2$ (in the sense of Definition 2) using $\mathcal{A}_1$ as a subroutine:

1. $\mathcal{A}_2$ is given pk, which it gives to $\mathcal{A}_1$.
2. When $\mathcal{A}_1$ queries the proof oracle, $\mathcal{A}_2$ answers the query using the zero-knowledge simulator.
3. When $\mathcal{A}_1$ requests a challenge nonce, $\mathcal{A}_2$ chooses a nonce nonce^* as described in Section 1.2 and gives it to $\mathcal{A}_1$.
4. If Succ occurs then let (c, s) be the proof output by $\mathcal{A}_1$ and let $A = g^s/\mathsf{pk}^c$. Then:

   > Continually rewind $\mathcal{A}_1$ to the point where it makes[4] the query $H(\mathsf{pk}, \mathsf{nonce}^*, A)$, returning a uniform response each time, until $\mathcal{A}_1$ again outputs a valid proof $\pi' = (c', s')$ for which $g^{s'}/\mathsf{pk}^{c'} = A$ and $c' \neq c$. (In parallel, $\mathcal{A}_2$ performs an exhaustive search for x.) Then extract x as described in Section 2.2 and output x.

One can show that the expected number of times $\mathcal{A}_2$ rewinds $\mathcal{A}_1$ is constant. Moreover, $\mathcal{A}_2$ will always output $x = \log_g \mathsf{pk}$ whenever Succ occurs. Thus, the probability with which $\mathcal{A}_2$ correctly computes x is exactly ϵ_3. □

The above allows us to focus our attention on Definition 2. In particular, if we can prove a bound on the success probability of attacks in the sense of Definition 2 then we can use that to derive a bound on the success probability of attacks in the sense of Definition 1.

3 Hardness of Guessing the Secret

As noted earlier, an attacker can succeed in the above experiment (regardless of the distribution $\mathcal{D}$) if it can guess sk or if it can solve the discrete-logarithm problem in $\mathbb{G}$. We consider these two possibilities independently.

For a distribution $\mathcal{D}$ over sk, define the *min-entropy* of $\mathcal{D}$ as

$$H_\infty(\mathcal{D}) = -\log \max_s \{\Pr[\mathsf{sk} = s]\}.$$

The min-entropy of § serves as a measure of how easy it is to guess sk when it is chosen according to $\mathcal{D}$. In particular, there is a strategy for guessing sk in a single guess that succeeds with probability $2^{-H_\infty(\mathcal{D})}$ (and this is optimal); the probability of guessing sk in q_H guesses is at most $q_H \cdot 2^{-H_\infty(\mathcal{D})}$. From this point of view, distributions with higher min-entropy are more secure for users in the sense that attackers are less likely to guess a user's secret.

For security of the identification scheme to hold, we also need to use a group in which the discrete-logarithm problem is hard. For a fixed cyclic group $\mathbb{G}$ of order q with generator g, and any algorithm $\mathcal{A}$, define

$$\mathsf{Adv}_{\mathsf{dlog}}(\mathcal{A}) = \Pr[x \leftarrow \mathbb{Z}_q; x' \leftarrow \mathcal{A}(g^x) : x' = x].$$

We say the discrete-logarithm problem is (t, ϵ)-hard (for this $\mathbb{G}$ and g) if for all $\mathcal{A}$ running in time at most t it holds that $\mathsf{Adv}_{\mathsf{dlog}}(\mathcal{A}) \leq \epsilon$.

The following theorem shows that these are the only ways to determine x:

[4] We can assume without loss of generality that $\mathcal{A}_1$ makes such a query.

Theorem 2. *Fix a distribution $\mathcal{D}$ and an adversary $\mathcal{A}$ running in time t and making q_H queries to H, where H is modeled as a random oracle. If the discrete-logarithm problem is (t, ϵ)-hard for $\mathbb{G}$ and g as used by Π, then $\mathsf{Adv}^2_{\mathcal{D}}(\mathcal{A}) \leq q_H \cdot 2^{-H_\infty(\mathcal{D})} + \epsilon$.*

Proof. Let $\delta = \mathsf{Adv}^2_{\mathcal{D}}(\mathcal{A})$. This is the probability with which $\mathcal{A}$ correctly outputs x in the experiment of Definition 2.

Consider next the modified experiment in which we choose sk according to $\mathcal{D}$, choose a uniform $x \in \mathbb{Z}_q$, set $\mathsf{pk} := g^x$, and give pk to $\mathcal{A}$. Then, for each of $\mathcal{A}$'s queries to H, we simply return a uniform value. Let δ' denote the probability with which $\mathcal{A}$ outputs x here. It is not hard to see that this modified experiment is identical to the experiment of Definition 2 unless $\mathcal{A}$ ever queries sk to H, which occurs with probability at most $q_H \cdot 2^{-H_\infty(\mathcal{D})}$. Thus, $\delta' \geq \delta - q_H \cdot 2^{-H_\infty(\mathcal{D})}$.

Finally, consider the modified experiment in which we are given an element $\mathsf{pk} \in \mathbb{G}$ that we pass to $\mathcal{A}$. Then, for each of $\mathcal{A}$'s queries to H, we simply return a uniform value. The probability that $\mathcal{A}$ outputs $x = \log_g \mathsf{pk}$ is exactly the same as in the previous experiment, i.e., δ'. However, by assumption regarding the hardness of the discrete-logarithm problem, we must have $\delta' \leq \epsilon$. The theorem follows. □