PhantomID - A Secure Anonymous Network

Nnamdi Michael Okpala

December 12, 2024

Abstract

We present PhantomID, a novel secure anonymous network protocol that implements a tree-based node structure with dynamic identity management. The system provides robust security guarantees through cryptographic verification, distributed trust, and automatic rebalancing mechanisms, enhanced by a dedicated security daemon. We demonstrate the protocol's security through formal proofs and analyze its resistance to common attack vectors.

1 Introduction

Let $\mathcal{N}$ represent our network with node set V and edge set E. We define the security parameters as follows:

- Let λ be the security parameter
- Let $H : \{0,1\}^* \to \{0,1\}^\lambda$ be a cryptographic hash function (SHA-256)
- Let $ID_{static} : V \to \{0,1\}^\lambda$ be the static ID assignment function
- Let $ID_{dynamic} : V \times T \to \{0,1\}^\lambda$ be the dynamic ID generation function
- Let T denote the set of timestamps

2 Network Initialization

2.1 Root Node Generation

For initial network setup at time t_0:

1. Generate root node $r \in V$
2. Compute root key: $k_r = H(r \parallel t_0)$
3. Initialize empty tree $\mathcal{T} = (V, E)$

2.2 Static ID Assignment

For each initial node $v_i \in V$:

$$ID_{static}(v_i) = H(k_r \parallel v_i \parallel t_0)$$

2.3 Root Destruction Protocol

After initialization:

1. Securely erase k_r using:

$$k_r \leftarrow k_r \oplus RNG(\lambda)$$

2. Verify: $\forall v_i \in V : Verify(ID_{static}(v_i)) = 1$

3 Node Join Protocol

For new node v_{new} at time t_j:

1. Generate static ID:

$$s_{new} = ID_{static}(v_{new}) = H(v_{new} \parallel t_j)$$

2. Calculate dynamic ID:

$$d_{new} = ID_{dynamic}(v_{new}, t_j) = H(s_{new} \parallel State_{\mathcal{T}} \parallel t_j)$$

where $State_{\mathcal{T}}$ represents the current tree state.

Algorithm 1 Node Join

```
1: procedure NodeJoin($v_{new}$, $t_j$)
2:     $s_{new} \leftarrow H(v_{new} \parallel t_j)$
3:     $state \leftarrow$ GetTreeState($\mathcal{T}$)
4:     $d_{new} \leftarrow H(s_{new} \parallel state \parallel t_j)$
5:     if VerifyUniqueness($d_{new}, \mathcal{T}$) then
6:         UpdateTree($\mathcal{T}, v_{new}, d_{new}$)
7:         RebalanceTree($\mathcal{T}$)
8:         return Success
9:     end if
10:    return Failure
11: end procedure
```

4 Node Departure Protocol

For departing node v_d at time t_d:

1. Locate node: $v_d \in V$ where $ID_{static}(v_d) = s_d$
2. Remove static ID: $V \leftarrow V \setminus \{v_d\}$
3. Update tree state: $State_{\mathcal{T}} \leftarrow H(State_{\mathcal{T}} \parallel t_d)$

Algorithm 2 Node Departure

```
1: procedure NodeDepart($v_d$, $t_d$)
2:     $s_d \leftarrow ID_{static}(v_d)$
3:     if VerifyNode($v_d, s_d$) then
4:         RemoveNode($\mathcal{T}, v_d$)
5:         $state \leftarrow H($GetTreeState$(\mathcal{T}) \parallel t_d)$
6:         UpdateTreeState($\mathcal{T}, state$)
7:         RebalanceTree($\mathcal{T}$)
8:         return Success
9:     end if
10:    return Failure
11: end procedure
```

5 Security Analysis

5.1 Security Properties

1. **Node Anonymity:**
   For any adversary $\mathcal{A}$ and nodes $v_i, v_j \in V$:

$$Pr[\mathcal{A}(ID_{dynamic}(v_i, t)) = v_i] \leq negl(\lambda)$$

2. **Tree Integrity:**
   For tree state $State_{\mathcal{T}}$ at time t:

$$Verify(State_{\mathcal{T}}, t) = H(\prod_{v \in V} ID_{static}(v) \parallel t)$$

5.2 Attack Resistance

Theorem 1 (Node Impersonation Resistance)**.** *Given security parameter λ, for any PPT adversary $\mathcal{A}$:*

$$Pr[\mathcal{A}(1^\lambda) \text{ produces valid } ID_{static}] \leq negl(\lambda)$$

Proof. By reduction to SHA-256 collision resistance. □

Algorithm 3 Tree Rebalancing

```
1: procedure RebalanceTree($\mathcal{T}$)
2:     $heights \leftarrow$ ComputeSubtreeHeights($\mathcal{T}$)
3:     while $\neg$IsBalanced($heights$) do
4:         $node \leftarrow$ FindHighestImbalance($\mathcal{T}$)
5:         RotateSubtree($\mathcal{T}, node$)
6:         UpdateTreeState($\mathcal{T}$)
7:         $heights \leftarrow$ ComputeSubtreeHeights($\mathcal{T}$)
8:     end while
9:     VerifyIntegrity($\mathcal{T}$)
10: end procedure
```

6 Phantom Daemon Architecture

6.1 Overview

The Phantom Daemon operates as a privileged system service that provides an isolated security layer for the PhantomID network. It implements the following key features:

- Privileged process isolation
- Network traffic encryption
- Identity management subsystem
- State verification service

6.2 Daemon Process Model

Let $\mathcal{D}$ represent the daemon process with privilege set P and isolation domain I. We define:

$$Daemon_{state} = \{(p, i) \in P \times I : Verify(p) \wedge Isolate(i)\}$$

where $Verify(p)$ confirms privilege legitimacy and $Isolate(i)$ ensures process isolation.

6.3 Security Layer Implementation

6.4 Layer Interface

The daemon provides a secure interface layer:

1. Network Protocol Interface:
$$\mathcal{I}_{net} : \mathcal{D} \times V \to \{0, 1\}^*$$
2. Identity Management Interface:
$$\mathcal{I}_{id} : \mathcal{D} \times ID_{static} \to ID_{dynamic}$$
3. State Verification Interface:
$$\mathcal{I}_{state} : \mathcal{D} \times State_{\mathcal{T}} \to \{0, 1\}$$

Algorithm 4 Daemon Initialization

```
1: procedure InitializeDaemon($config$)
2:     $privileges \leftarrow$ RequestPrivileges()
3:     $isolation \leftarrow$ CreateIsolationDomain()
4:     $daemon\_state \leftarrow (privileges, isolation)$
5:     if VerifyDaemonState($daemon\_state$) then
6:         StartSecurityLayer($daemon\_state$)
7:         InitializeNetworkEncryption()
8:         return Success
9:     end if
10:    return Failure
11: end procedure
```

6.5 Security Guarantees

Theorem 2 (Daemon Isolation). *For any unprivileged process $p \notin P$:*

$$Pr[p \text{ accesses } \mathcal{D}] \leq negl(\lambda)$$

Proof. By reduction to operating system process isolation guarantees. □

Algorithm 5 Daemon Security Layer

```
1: procedure ProcessNetworkRequest($request$, $node$)
2:     $valid \leftarrow$ ValidateRequest($request$)
3:     if $valid$ then
4:         $encrypted \leftarrow$ EncryptPayload($request.data$)
5:         $signature \leftarrow$ SignRequest($encrypted$)
6:         TransmitSecure($node, encrypted, signature$)
7:         return Success
8:     end if
9:     return Failure
10: end procedure
```

7 Implementation Considerations

7.1 Thread Safety Protocol

For concurrent operations:

1. Lock acquisition: $\mathcal{L}(op) = \{l_1, ..., l_k\}$
2. Operation execution: $Execute(op, State_{\mathcal{T}})$
3. Lock release: $\mathcal{R}(op) = \emptyset$

Algorithm 6 Integrity Verification

```
1: procedure VerifyIntegrity($\mathcal{T}$)
2:     $checksum \leftarrow 0$
3:     for all $node\ v \in \mathcal{T}$ do
4:         $static\_id \leftarrow ID_{static}(v)$
5:         $dynamic\_id \leftarrow ID_{dynamic}(v, current\_time)$
6:         $checksum \leftarrow H(checksum \parallel static\_id \parallel dynamic\_id)$
7:     end for
8:     return $checksum = \textsc{GetTreeState}(\mathcal{T})$
9: end procedure
```

8 Security Recommendations

1. Regular key rotation:
   $\forall v \in V : ID_{dynamic}(v, t_{new}) \leftarrow H(ID_{dynamic}(v, t_{old}) \parallel t_{new})$
2. Integrity verification frequency:
   Minimum interval $\Delta t = 300s$
3. Maximum tree depth:
   $depth(\mathcal{T}) \leq \log_2(|V|) + c$, where c is a small constant
4. Daemon security requirements:
   - Mandatory process isolation
   - Encrypted inter-process communication

- Regular privilege verification
- Protected memory space $\geq$ 256MB

9 Conclusion

PhantomID provides provable security guarantees while maintaining efficient network operations. The protocol's resistance to common attacks is mathematically verified, and its implementation considerations ensure practical deployability. The addition of the Phantom Daemon layer provides enhanced security through privileged process isolation and secure communication channels.

References

[1] Bellare, M., & Rogaway, P. (1993). *Random oracles are practical: A paradigm for designing efficient protocols.*

[2] Merkle, R. C. (1987). *A digital signature based on a conventional encryption function.*

[3] NIST. (2015). *SHA-256 Standard: Secure Hash Standard.*