



A Deep Dive into the Nym Mix Network Audit

Mario Heiderich, Alexander Pirker,
Daniel Bleichenbacher, Luan
Herrera, Marta Conde, **Nadim
Kobeissi**



About **CURE+53**

- Auditing team largely based in the EU.
- ~25 core members as of 2025.
- Probably over 1,000 audits at this point.
- This audit was co-authored with:
 - Alexander Pirker
 - Daniel Bleichenbacher
 - Luan Herrera
 - Marta Conde

Audit Timeline

Identified Vulnerabilities

[NYM-01-008 WP5: eCash vulnerable to unintended payInfo collisions \(Low\)](#)
[NYM-01-009 WP5: BLS12-381 EC signature bypasses in Coconut library \(Critical\)](#)
[NYM-01-014 WP5: Partial signature bypass in offline eCash \(Critical\)](#)
[NYM-01-016 WP2: Hard-coded "fast nodes" influence traffic distribution \(Low\)](#)
[NYM-01-020 WP3: Replaying Sphinx packets in mixnet could facilitate DoS \(Low\)](#)
[NYM-01-024 WP1: Credentials and key material insecurely stored in iOS \(Medium\)](#)
[NYM-01-027 WP3: Nonce-key reuse in AES-CTR in Nym gateways \(Critical\)](#)
[NYM-01-030 WP3: Gateway skips credential serial number check \(Critical\)](#)
[NYM-01-032 WP3: Bloom filter parameters yield false positives \(High\)](#)
[NYM-01-033 WP5: Signature forgery of Pointcheval-Sanders scheme \(Critical\)](#)
[NYM-01-034 WP3: Nym network monitors have no persistent identity \(Medium\)](#)
[NYM-01-042 WP5: Faulty aggregation to invalid offline eCash signatures \(Critical\)](#)

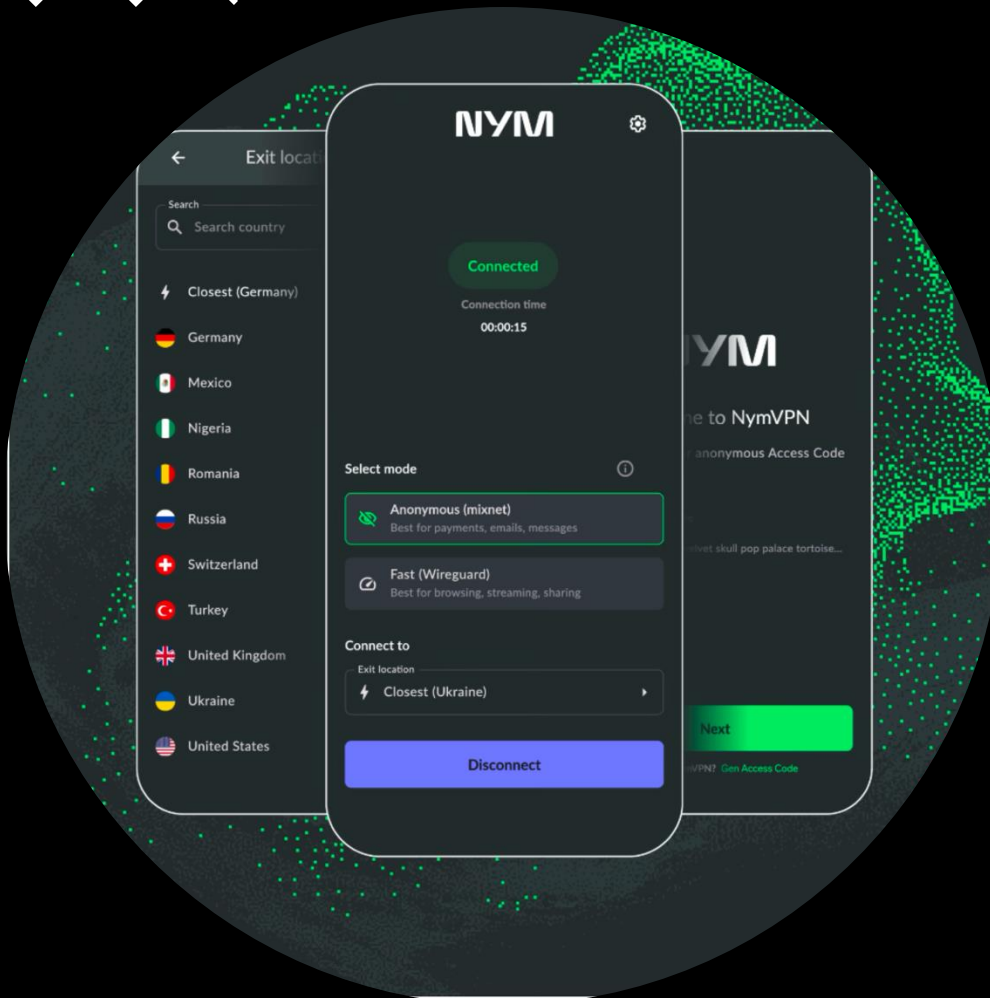
Miscellaneous Issues

[NYM-01-001 WP3: Bloom filter migration to Binary Fuse filters \(Low\)](#)
[NYM-01-002 WP5: Constant zero nonces in AES-CTR for Sphinx protocol \(Low\)](#)
[NYM-01-003 WP5: Panics in Sphinx protocol due to short packets \(Medium\)](#)
[NYM-01-004 WP1: Android app supports unmaintained SDK versions \(Low\)](#)
[NYM-01-005 WP5: No infinity point check reveals plaintext for ElGamal \(High\)](#)
[NYM-01-006 WP5: Collisions in hash values of Coconut challenges \(Low\)](#)
[NYM-01-007 WP5: Verification of KappaZeta NIZKP succeeds for ju](#)
[NYM-01-010 WP1: Android / iOS apps lack root / jailbreak detection](#)
[NYM-01-011 WP1: Absent security screen in apps facilitates](#)

- **July 2024:** Audit conducted on:
 - Nym Cryptographic Components
 - Nym Infrastructure
 - NymVPN Mobile & Desktop apps
- **January 2025:** Audit report published (at Nym's request)
- **March 2025:** Audit report temporarily removed (pending edits)

About Nym

- Innovative mixnet-based anonymous VPN.
- First mixnet in production.
- Uses decentralized mix network to provide anonymity (kind of similar to Tor but with more security guarantees).
- Mix nodes volunteer and mine cryptocurrency rewards by relaying traffic.
- New product: NymVPN (wrapper for consumer devices).



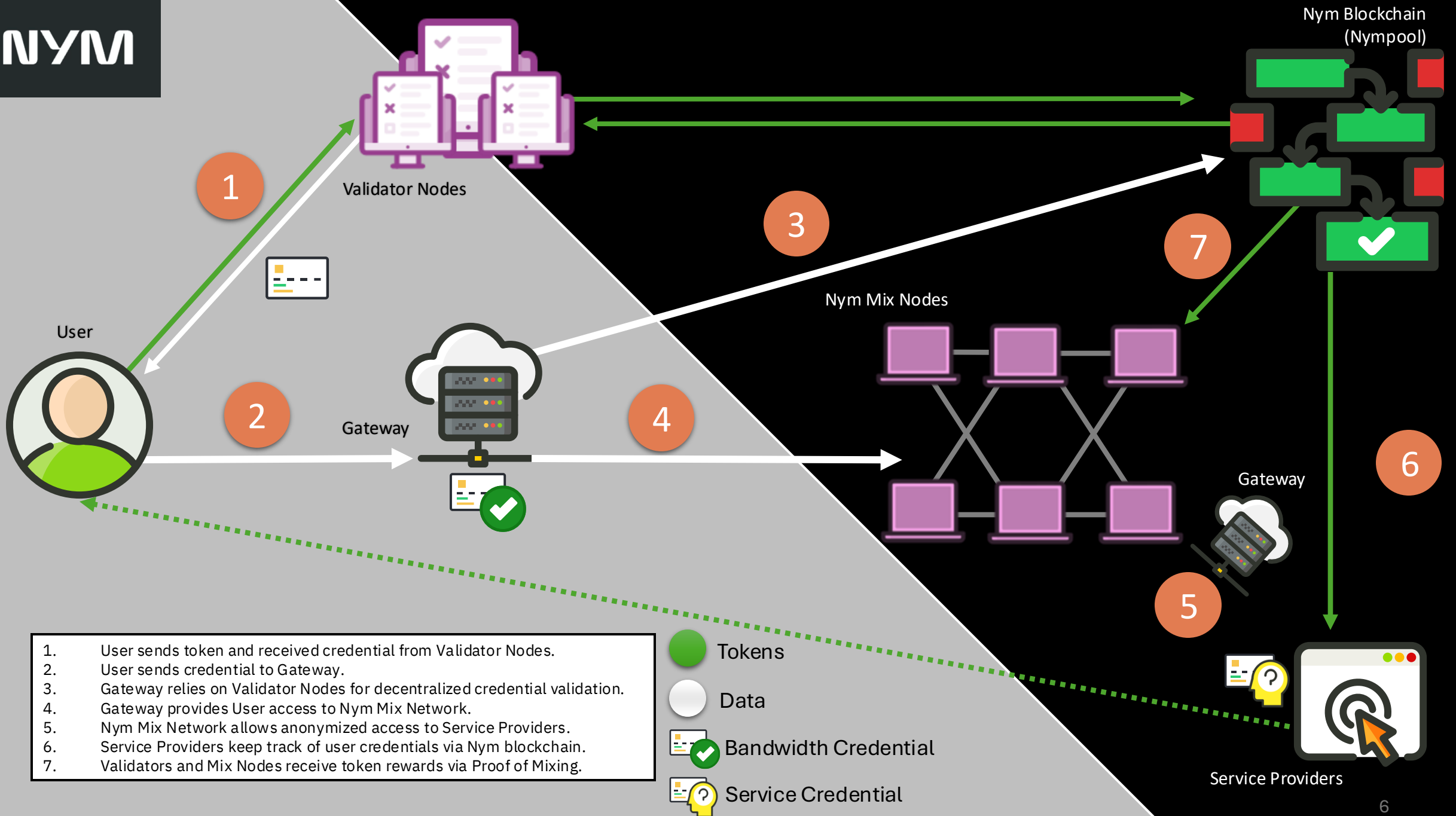
Onion Routing vs. Mix Network

Onion Routing (Tor)





- Uses layered encryption, routing traffic through multiple nodes.
- Emphasizes real-time communication.
- Packets follow a fixed circuit for a short-lived session.

Mix Networks (Nym)

- Uses “mixing”, where packets from many users are collected, reordered and delayed before forwarding.
- Emphasizes high-level anonymity through statistical obfuscation, introducing latency to defeat stronger adversaries.
- Higher latency by design.



1. User sends token and received credential from Validator Nodes.
2. User sends credential to Gateway.
3. Gateway relies on Validator Nodes for decentralized credential validation.
4. Gateway provides User access to Nym Mix Network.
5. Nym Mix Network allows anonymized access to Service Providers.
6. Service Providers keep track of user credentials via Nym blockchain.
7. Validators and Mix Nodes receive token rewards via Proof of Mixing.

-  Tokens
-  Data
-  Bandwidth Credential
-  Service Credential

Many cryptographic components!

All kinds of signature schemes.

Anonymous credentials.

Symmetric encryption.

Public key cryptography.

Statistical access control.

Blockchains, validator nodes, etc.

Custom packet formats.

...to name a few.

This Talk

- Interesting cryptographic findings and attacks.
- Showcasing evolving types of vulnerabilities in cutting-edge in-production cryptographic software in 2025.
- Not all the vulnerabilities!
 - “Best-of”: 10 out of 43 issues.


```

    let invalid = BlindedSignature(G1Projective::identity(),
G1Projective::identity());
    let invalid_priv_commitments = vec![G1Projective::identity(),
G1Projective::identity()];
    assert!(verify_partial_blind_signature(
        &params,
        &invalid_priv_commitments,
        &public_attributes,
        &invalid,
        validator_keypair.verification_key()
    ));

    random_scalars_refs!(&public_attributes1, params, 3);
    assert!(verify_partial_blind_signature(
        &params,
        &invalid_priv_commitments,
        &public_attributes1,
        &invalid,
        validator_keypair.verification_key()
    ));
}

```

NYM

“We do verifications upstream in the codebase; while it affects the Coconut library, it doesn’t affect our stack”

NYM-01-009: BLS12-381 EC Signature Bypass in Coconut Library

- Coconut: “distributed cryptographic signing scheme”.
- Attacker is able to provide bogus credentials together with invalid signatures to trick rewarder.
- **Fix:** validate signature inputs for infinity points etc. on BLS12-381.

```

fn successful_verify_partial_blind_signature_infinity_points() {
    let invalid = BlindedSignature {
        h: G1Projective::identity(),
        c: G1Projective::identity(),
    };
    let keys = ttp_keygen(2, 3).unwrap();
    let private_attributes = vec![G1Projective::identity(),
G1Projective::identity()];
    random_scalars_refs!(public_attributes, ecash_group_parameters(),
3);
    assert!(!verify_partial_blind_signature(
        &private_attributes,
        &public_attributes,
        &invalid,
        &keys[0].verification_key(),
    ));
}

```

NYM-01-014: Partial Signature Bypass in Offline eCash

- Lack of signature validation also extended to Nym’s “offline eCash” anonymous credentials scheme.
- **Fix:** validate signature inputs for infinity points and other invalid inputs.



“We do verifications upstream in the codebase; while it affects the offline eCash library, it doesn’t affect our stack”

Compute the forged signature for $a_1=a/2$, $a_2=a/2$:

$$\begin{aligned} S_1' &= \text{hash_to_g1}((a_1 + a_2) * G_1) \\ &= \text{hash_to_g1}((a/2 + a/2) * G_1) \\ &= \text{hash_to_g1}(a * G_1) = S_1 = S_1' = H \end{aligned}$$

$$\begin{aligned} S_2' &= (1/2) * S_2 + (1/2) * S_2' = (x/2 + a/2 * y_1) * H + (x/2 + a/2 * y_2) * H \\ &= (x + a/2 * y_1 + a/2 * y_2) * H \end{aligned}$$



“Risk does not apply upstream since Coconut and offline eCash do not uniquely use public attributes for H ”

NYM-01-033: Signature Forgery of Pointcheval-Sanders Scheme

- Original Pointcheval-Sanders paper requires H to be random for each signing operation.
- Nym (Coconut) uses the sum of public attributes when deriving H .
- To forge a signature for $[a/2, a/2]$, attacker only needs signatures for $[a, 0]$ and $[0, a]$.
- **Fix:** Ensure unique H .

```
pub fn check_bilinear_pairing(p: &G1Affine, q: &G2Prepared, r: &G1Affine,
s: &G2Prepared) -> bool {
    // checking e(P, Q) * e(-R, S) == id
    // is equivalent to checking e(P, Q) == e(R, S)
    // but requires only a single final exponentiation rather than two of
    them
    // and therefore, as seen via benchmarks.rs, is almost 50% faster
    // (1.47ms vs 2.45ms, tested on R9 5900X)

    let multi_miller = multi_miller_loop(&[(p, q), (&r.neg(), s)]);
    multi_miller.final_exponentiation().is_identity().into()
}
```

NYM-01-042: Faulty Aggregation to Invalid Offline eCash Signatures

- Signature aggregation function accepts $(\infty,)$ and $(\infty, -)$, produces (∞, ∞) .
- **Fix:** once again, validate inputs to prevent points at infinity, identity points, etc.



“We do verifications upstream in the codebase; while it affects the offline eCash library, it doesn’t affect our stack”

```

pub fn encrypt_and_tag(
    &self,
    data: &[u8],
    iv: Option<&IV<GatewayEncryptionAlgorithm>>,
) -> Vec<u8> {
    let encrypted_data = match iv {
        [...]
        None => {
            let zero_iv =
stream_cipher::zero_iv::<GatewayEncryptionAlgorithm>();
            stream_cipher::encrypt::<GatewayEncryptionAlgorithm>(
                self.encryption_key(),
                &zero_iv,
                data,
            )
        }
    };
    [...]
}
[...]
pub fn encryption_key(&self) -> &CipherKey<GatewayEncryptionAlgorithm> {
    &self.encryption_key
}

```

NYM

“Fixing the issue and upgrading to AES-GCM-SIV as recommended.”

NYM-01-027: Nonce-Key Reuse in AES-CTR in Nym Gateways

- Nym gateways encrypt data using AES-CTR with a fixed nonce of 0, and a unique, non-rotating key.
- $C_1 \oplus C_2 = M_1 \oplus M_2$
- **Fix:** don't use a fixed nonce, rotate keys, switch to authenticated AES mode such as AES-GCM-SIV.

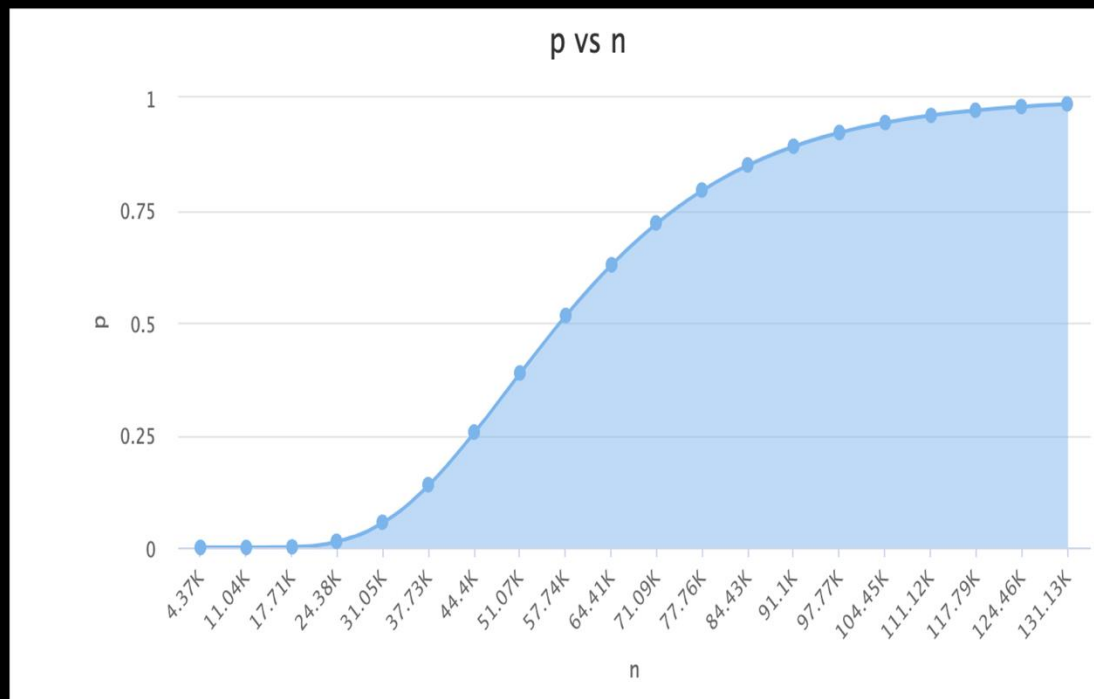
- **Vendor separation:** By incorporating a unique *vendorId* for each vendor, the scheme ensures that transactions from different vendors will always produce different identifiers, even if the *payInfo* is identical. This significantly reduces the risk of cross-vendor collisions.
- **Context-specific identifiers:** The inclusion of a context parameter allows for further differentiation of transactions. This could be used to separate different types of transactions, or to handle cases where the same payment information might be used in different contexts.
- **Improved uniqueness:** *HKDF* is designed to generate multiple keys from a single input key material. It's particularly well-suited for deriving unique, cryptographically strong identifiers.
- **Better resistance to attacks:** The use of *HKDF* makes it more difficult for an attacker to manipulate or predict transaction identifiers, as they would need to know the *vendorId* and *context* in addition to the *payInfo*.
- **Scalability:** As the system grows and more vendors are added, the *HKDF* approach continues to provide strong guarantees of uniqueness and separation between vendors.

The Nym logo, consisting of the letters 'NYM' in a bold, white, sans-serif font, centered within a solid blue square.

“Acknowledged, will address in future release”

NYM-01-008: eCash Vulnerable to Unintended *PayInfo* Collisions

- Nym uses “offline eCash with threshold issuance” to issue redeemable credentials for NymVPN usage allowance.
- $H(\text{payInfo})$ is meant to generate a unique identifier for each payment. However, no namespacing for vendor vs. individual-payment information.
- **Fix:** Derive hashes using $\text{HKDF}(\text{vendorId}, \text{context}, \text{payInfo})$



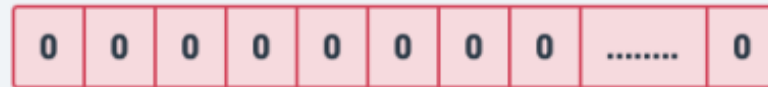
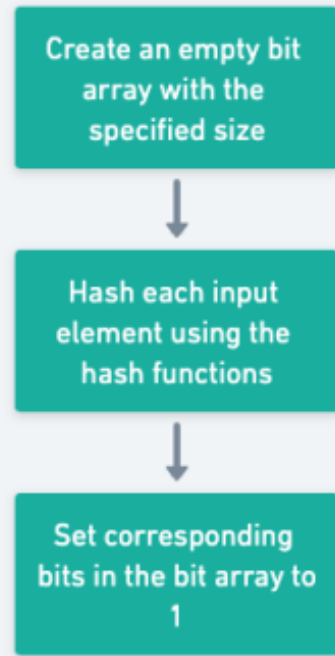
NYM-01-030: Gateway Skips Credential Serial Number Check

- Nym credentials have a serial number that prevents duplicate use.
- Storing all used serial numbers doesn't scale.
- Bloom filters are implemented, but the check doesn't actually happen.
- **Fix:** make the check happen.

NYM

“Will adopt fix and upgrade to Binary Fuse Filters”

Bloom Filter



Fixed size bit array, suppose n is length

$K=3$ Hash functions

$hash1(input1) \rightarrow x \% n = 3$,
where x is output and n is length of array

$hash3(input1) \rightarrow z \% n = 13$,
where z is output and n is length of array

$hash2(input1) \rightarrow y \% n = 5$,
where y is output and n is length of array

$hash1(input1) \rightarrow x \% n = 3$,
where x is output and n is length of array

$hash3(input1) \rightarrow z \% n = 13$,
where z is output and n is length of array

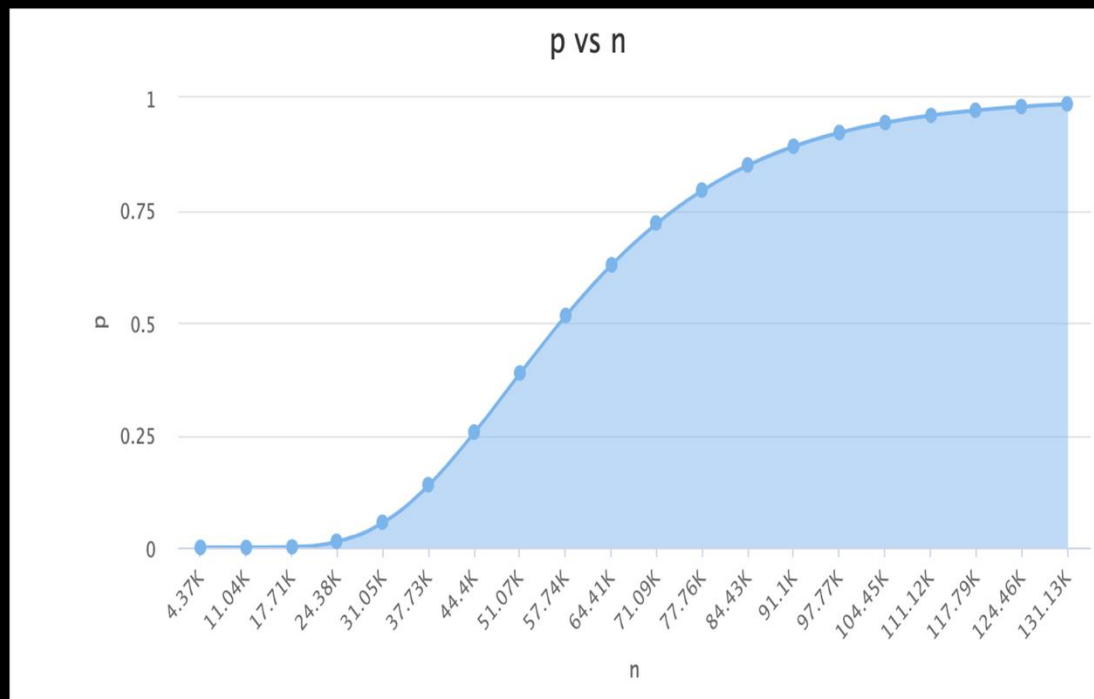
$hash2(input1) \rightarrow y \% n = 5$,
where y is output and n is length of array

at index 3

at index 13

at index 5





NYM-01-032: Bloom Filter Parameters Yield False Positives

- Bloom Filter parameters are insufficient, would yield a 1 in 6 false positive rate after only 40,000 entries, or 2,000 Nym users (roughly).
 - $K = 13$
 - $M = 250,000$
- **Fix:** improve parameters.

NYM

“Will adopt fix and upgrade to Binary Fuse Filters”

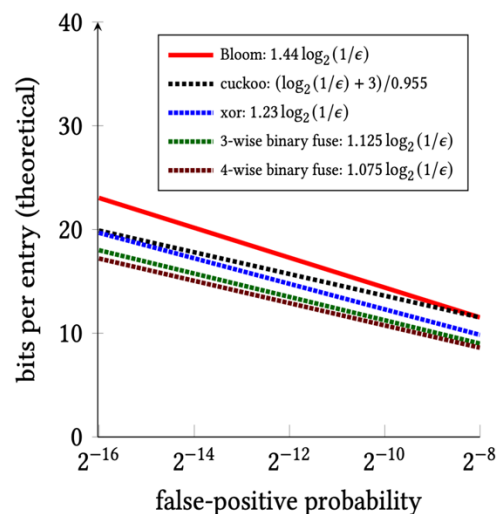


Fig. 1. Theoretical memory usage for Bloom filters (optimized for space), cuckoo filters (at maximal capacity [11, Table 2]), xor filters and binary fuse filters given a desired bound on the false-positive probability from 2^{-16} to 2^{-8} .

NYM-01-032: Bloom Filter Parameters Yield False Positives

- Bloom Filter parameters are insufficient, would yield a 1 in 6 false positive rate after only 40,000 entries, or 2,000 Nym users (roughly).
 - $K = 13$
 - $M = 250,000$
- **Fix:** improve parameters.

NYM

“Will adopt fix and upgrade to Binary Fuse Filters”

```
if (cmd === 'get_fastest_node_location') {  
  return new Promise<Country>((resolve) =>  
    resolve({  
      name: 'France',  
      code: 'FR',  
    })),  
  );  
}
```

The Nym logo, consisting of the letters 'NYM' in a bold, sans-serif font, with the 'Y' and 'M' joined together.

“Fixed in production builds”

NYM-01-016: Hard-Coded “Fast Nodes” Influence Traffic Distribution

- NymVPN desktop client hard-coded France as the “fastest mixnet country” on the front-end.
- Affects traffic distribution, may negatively impact anonymity set.
- **Fix:** implement dynamic, real-time system for determining fastest nodes based on network conditions.

Ed25519 private key (in *private_identity.pem*):

```
-----BEGIN ED25519 PRIVATE KEY-----
AG74IiHF5PMrQlhFZWHVGhL7of29/ohyn+EKM8Uoz0s=
-----END ED25519 PRIVATE KEY-----
```

X25519 private key (in *private_encryption.pem*):

```
-----BEGIN X25519 PRIVATE KEY-----
+08nOWIwCTp1trygefBzQIyUokz46Gi0IxBod8B5nSU=
-----END X25519 PRIVATE KEY-----
```

List of files:

```
-rw----- 1 mobile mobile 45056 Jul  5 11:40 credentials_database.db
-rw----- 1 mobile mobile  116 Jul  5 11:40 public_identity.pem
-rw----- 1 mobile mobile  118 Jul  5 11:40 private_identity.pem
-rw----- 1 mobile mobile  114 Jul  5 11:40 public_encryption.pem
-rw----- 1 mobile mobile  116 Jul  5 11:40 private_encryption.pem
-rw----- 1 mobile mobile  124 Jul  5 11:40 ack_key.pem
-rw-r--r-- 1 mobile mobile 49152 Jul 15 19:36 gateways_registrations.sqlite
-rw-r--r-- 1 mobile mobile 61440 Jul 15 19:36 persistent_reply_store.sqlite
```

NYM-01-024: Credentials and Key Material Insecurely Stored on iOS

- NymVPN app stored all private credentials unencrypted on iOS devices. Retrievable via file dump.
- **Fix:** Use iOS Keychain API with *AfterFirstUnlockThisDeviceOnly*.



“Will be fixed in a future release”

```
fn perform_initial_packet_processing(
    &self,
    packet: NymPacket,
) -> Result<NymProcessedPacket, MixProcessingError> {
    nanos!("perform_initial_packet_processing", {
        packet.process(&self.sphinx_key).map_err(|err| {
            debug!("Failed to unwrap NymPacket packet: {err}");
            MixProcessingError::NymPacketProcessingError(err)
        })
    })
}
```

Furthermore, the excerpt below demonstrates the start of processing a new Sphinx packet. It is clear that the mixnet node connection handler forwards framed Sphinx packets to the processor without deduplication.

Affected file #2:

nym/mixnode/src/node/listener/connection_handler/mod.rs

Affected code #2:

```
fn handle_received_packet(&self, framed_sphinx_packet: FramedNymPacket) {
    //
    // TODO: here be replay attack detection - it will require similar key
    // cache to the one in
    // packet processor for vpn packets,
    // question: can it also be per connection vs global?
    //
    [...]
    nanos!("handle_received_packet", {
        match self.packet_processor.process_received(framed_sphinx_packet) {
            Err(err) => debug!("We failed to process received sphinx packet -
            {err}"),
            Ok(res) => match res {
                MixProcessingResult::ForwardHop(forward_packet, delay) => {
                    self.delay_and_forward_packet(forward_packet, delay)
                }
                MixProcessingResult::FinalHop(..) => {
                    warn!("Somehow processed a loop cover message that we
                    haven't implemented yet!")
                }
            },
        }
    })
}
```

NYM-01-020: Replaying Sphinx Packets In Mixnet Could Facilitate DoS

- Rogue mixnet node can replay existing packets within the Nym mixnet.
- **Fix:** rotate Sphinx keys on a regular basis, detect replayed Sphinx packets.

NYM

“Will be fixed in a future release”

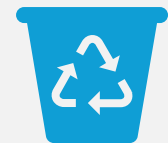
Resolution



Nym worked on addressing bugs, provided feedback for report publication.



Fixing findings from a strong audit is how products mature from R&D to consumer-grade.

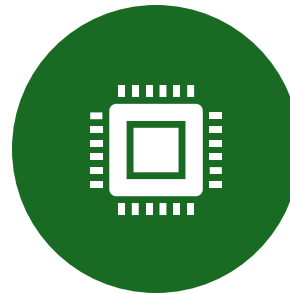


Producing reusable components is a good thing.

Takeaways



More complex cryptography software is seeing largely **old bugs in new constructions**.



AES-CTR nonce reuse: ancient vulnerability.



Validation in esoteric signature schemes: old vulnerability in a different primitive.



Anonymous credentials, custom packet formats: actual new stuff.



Thank you!

- **Cure53:** <https://cure53.de>
- **Personal:** <https://nadim.computer>

With thanks to my teammates:

- Alexander Pirker
- Daniel Bleichenbacher
- Luan Herrera
- Marta Conde