Cryptographic Algorithms and Security Protocols for ICN

Christopher A. Wood
UC Irvine, PARC
woodc1@uci.edu, cwood@parc.com
ICN Network Stack

native

overlay

(IP) App

ICN

TCP/UDP

IP

Link

PHY
Q: What crypto and security techniques can enable these core services in the network?
Selection Criteria

• Goal: Crypto that **can enables essential network services**

• What’s essential?
  – Integrity and authenticity
  – “Privacy” (e.g., as alluded to in RFC6973)
  – Availability

• What’s non essential?
  – Anonymity
  – Confidentiality (application-layer concern)
Topic Breakdown

- **Included**
  - Integrity and authenticity
    - Hash-based signatures
  - Privacy
    - Encrypted Deep Packet Inspection (DPI)
    - Password-Authenticated and Non-Interactive Key Exchange (PAKE and NIKE)
    - Private Information Retrieval (PIR)
    - Randomizable public-key encryption
    - Secure searchable encryption and predicate encryption (SSE and PE)
  - Availability
    - Authenticated Denial of Existence (DoE)

- **Omitted**
  - Oblivious and onion routing
  - Oblivious RAM
  - Fully and partially homomorphic encryption
  - Functional encryption (e.g., ABE and IBE)
Integrity and Authentication
Hash-Based Signatures

• Traditional signature schemes are based on trapdoor functions
  – RSA, DSA, ECDSA, etc.
  – PQ-secure?

• Hash-based signatures are quantum-secure, e.g., they don’t fall to Shor’s algorithm

• Based on one-time signatures (OTGs)
  – A key pair can be used only once
Lamport OTS Idea

Private key: \((x, y)\)

Public key: \((H(x), H(y))\)
Lamport OTS Idea

Private key: (x, y)

Public key: (H(x), H(y))

To sign a ‘0’: release x
To sign a ‘1’: release y
Merkle Tree Idea

H(H(AB)H(CD))

H(AB)  H(CD)

A  B  C  D
Merkle Tree Idea

\[ H(H(AB)H(CD)) \]

- Public key
- Parts of the public key

Public key:

- \( H(AB) \)
- \( H(CD) \)

Parts of the public key:

- \( k_A \) for node A
- \( k_B \) for node B
- \( k_C \) for node C
- \( k_D \) for node D
Merkle Tree Signature

Borrowed from:
https://cryptoservices.github.io/quantum/2015/12/07/many-times-signatures.html
XMSS Tree (Buchmann et al.)

Key (-pair):

- Height $h$
- $j = 0$

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash</td>
<td>Hash</td>
<td>Hash</td>
<td>Hash</td>
</tr>
<tr>
<td>Public key</td>
<td>Public key</td>
<td>Public key</td>
<td>Public key</td>
</tr>
</tbody>
</table>

XMSS:

- Root
- Authentication Path
- Active Key

Borrowed from:
http://www.square-up.org/index/hbs.html
Stateless Hash-Based Signature

• Similar to Merkle trees but the “index” is chosen at random

• Requires huge trees to avoid collisions
  – For security parameter $\lambda=128$ we require $2^{2\lambda}$ leaves (by birthday paradox)

• OTS secret keys are generated from PRGs
SPINCS (Bernstein et al.)

- Stateless hash-based signature with $2^{128}$ security
- Dramatically reduces signature and tree size
  - Signatures are down to 41KB
  - Public and private keys are 1KB
- Replace one-time leaves with “few-time leaves”
- Use XMSS-like per-node masks
Hash-Based Signature Recap

• Useful for long-term integrity in a post-quantum world
  – Will ICN data packets live forever?

• SPHINCS can be used as a drop-in replacement for current ICN signatures
Privacy
Encrypted DPI

• Goal: perform deep-packet-inspection (DPI) on encrypted packet payloads
  – e.g., to determine when a packet contains an encrypted version of a specific keyword

• Different measures of privacy
  – Exact-match privacy: only discover bytes that match target keywords
  – Probable cause privacy: decrypt a flow (entire packet) only if a keyword match is detected
BlindBox (Sheery et al.)

Borrowed from: 
BlindBox Details

• Packets are tokenized into tokens $t_1, \ldots, t_k$
• For each token $t$, encrypt as follows
  $$\text{salt}, AES_{AES-k(t)}(\text{salt})$$
• To detect a token $t$, precompute salt and token combinations
• Speedup: use a single salt per token and then derive subsequent token encryptions based on a counter

ensures randomness of identical tokens
BlindBox Rule Preparation

• The middlebox must learn AES-k(t) without revealing the encryption key k
• Solution: Garbled circuit to compute AES-k(t) for each token t
Encrypted DPI Recap

• ICN packets reveal information about producers and consumers
  – Signatures, names, etc.
• Carrying ICN packets in secure contexts can protect this information
• But how do we route on encrypted packets? Without trusting routers?
Privacy: Key Exchange
Password AKE

• Goal: create ephemeral keys from shared secrets – passwords (read: name) – in a way that:
  – Is not susceptible to offline dictionary attacks
  – Is forward secure
  – Does not require a PKI
• Varying number of rounds
  • Most are multi-round, e.g., J-PAKE (Hao et al.)
  • Some work in a **single round** without sacrificing forward secrecy (Benhamouda et al.)
Non-Interactive Key Exchange (NIKE)

- Goal: Two parties with knowledge of each other’s public keys agree on a shared secret without requiring any interaction

Alice: $x, g^x$
Bob: $y, g^y$
Shared key: $H(\text{“Alice”}, \text{“Bob”}, g^{xy})$

- Use case: WSN shared key derivation
- Fun fact: public-key encryption follows from NIKE
NIKE Protocols

• Setup: generate system parameters
• KeyGen: generate a private and public key pair for a given identity
• SharedKey: given (1) an identity, (2) its public key, (3) another identity, (4) and its secret key, compute and output a shared key
NIKE Protocols

• Setup: generate system parameters
• KeyGen: generate a private and public key pair for a given identity
• SharedKey: given (1) an identity, (2) its public key, (3) another identity, (4) and its secret key, compute and output a shared key

What’s in an ICN certificate?...
Forward Secure NIKE?

• Yes – it’s possible: evolve the public keys (Pointcheval et al.)
• Add an Update function to the scheme:
  – Move the secret key *forward* in time and space
• Use *multilinear* maps and a tree-based key derivation technique (described later)
KE Recap

• End-to-end sessions break the ICN model
• NIKE lets us create group “sessions” without any exchanges
• ICN entities (prefixes) have identities
Privacy: PIR
Private Information Retrieval

- A protocol to hide client requests to a server (e.g., a database)
- Two variants: computational (CPIR) and information-theoretic (ITPIR)
- CPIR (Stern and others)
  - Require only a single server
  - Much more computationally expensive
- ITPIR (Chor et al.)
  - Require at least two non-colluding servers
  - XOR-based computation but requires more communication
3) decode

1) query
2) answer
1) query
2) answer
ITPIR

C. Tschudin, Private Information Retrieval over ICN, INFOCOM 2016 NOM Workshop, April, 2016. [more later]
Privacy: Randomizable Encryption
Forward-Secure Public Key Encryption

• In plain English, “key-evolving public key encryption”

• Based on binary tree encryption
Binary Tree Encryption (Canetti et al.)

Root key pair: PK, SK_ε
Binary Tree Encryption (Canetti et al.)

Root key pair: \( PK, SK_\varepsilon \)

Encrypt with the public key

Diagram:
- Root node 0 with \( SK_0 \)
- Child node 1 with \( SK_1 \)
- Child node 2 with \( SK_2 \)
- Child node 3 with \( SK_3 \)
- Child node 4 with \( SK_4 \)
- Child node 5 with \( SK_5 \)
- Child node 6 with \( SK_6 \)
Binary Tree Encryption (Canetti et al.)

Root key pair: $PK, SK_\epsilon$

Encrypt with the public key

$SK_2$ derived from $SK_0$ and $PK$

$SK_0$

$SK_1$

$SK_3$

$SK_4$

$SK_5$

$SK_6$
Binary Tree Encryption (Canetti et al.)

Root key pair: PK, SK_ε

Encrypt with the public key

SK_2 derived from SK_0 and PK

Decrypt with SK_2

SK_0

0

SK_1

1

SK_3

3

SK_4

4

SK_5

5

SK_6

6

SK_2

2

Decrypt
Binary Tree Encryption (Canetti et al.)

Root key pair: \( PK, SK_\epsilon \)

Encrypt with the public key

\( SK_2 \) derived from \( SK_0 \) and \( PK \)

Decrypt with \( SK_2 \)

Evolve secret keys in time (and space)
Randomizable Public-Key Encryption

• Many encryption schemes can be re-randomized, e.g., ElGamal, BGN, etc.

• How can we maintain integrity after randomizing ciphertext?
Randomizable Public-Key Encryption

• Many encryption schemes can be re-randomized, e.g., ElGamal, BGN, etc.

• How can we maintain integrity after randomizing ciphertext?
  – Randomizable signatures
Randomizable Signatures (Blazy et al.)

- Given a ciphertext, anyone can re-randomize the ciphertext and adapt the signature the new encryption, i.e.,

\[ D_0 = \{ r' \leftarrow \mathcal{R}_e; s' \leftarrow \mathcal{R}_s : (c' = \text{Encrypt}(pk, vk, m; r'), \sigma' = \text{Sign}(sk, pk, c'; s')) \} \]

\[ D_1 = \{ r' \leftarrow \mathcal{R}_e; s' \leftarrow \mathcal{R}_s : (c', \sigma') = \text{Random}(vk, pk, c, \sigma; r', s') \} \]

are statistically indistinguishable
Extractable Signatures

\[ M \rightarrow \text{Encrypt}_{SC} \rightarrow C \]
\[ \text{Decrypt}_{\varepsilon} \rightarrow \sigma(M) \rightarrow \sigma(C) \]

Borrowed from:
Mixing in Motion

V

M

C^3, σ^3

Mixer (randomize)

C^2, σ^2

Mixer (randomize)

C^1, σ^1

S

decrypt and verify

encrypt and sign

M
Mixing in Motion

V

M

decrypt and verify

C³,σ³

Replica (randomize)

V

C²,σ²

Replica (randomize)

S

M

encrypt and sign

C¹,σ¹

V

M

decrypt and verify

C³,σ³

Replica (randomize)

V

C²,σ²

Replica (randomize)

S

M

encrypt and sign

C¹,σ¹

V

M

decrypt and verify

C³,σ³

Replica (randomize)

V

C²,σ²

Replica (randomize)

S

M

encrypt and sign

C¹,σ¹
Privacy: SSE and PE
Searchable Symmetric Encryption

• Goal: search over encrypted data using symmetric-key cryptography
• Given a database of (encrypted) documents and list of keywords, identify the documents that contain keywords
Searchable Symmetric Encryption

• Goal: search over encrypted data using symmetric-key cryptography

• Given a database of (encrypted) documents and list of keywords, identify the documents that contain keywords

• Many variants:
  – Interactive and non-interactive
  – Response-revealing and response-hiding
Efficient SSE (Song et al.)

\[ X_i = E(w_i) \]

\[ L_i \quad R_i \]

\[ S_i \quad F_{k_i}(S_i) \]

\[ C_i \]

\[ w_i \]

\[ F_{k_i} \]

pseudorandom value

Borrowed from:
http://eprints.eemcs.utwente.nl/24788/01/a18-bosch.pdf
# Other SSE Solutions

<table>
<thead>
<tr>
<th>Properties</th>
<th>[35, 25]</th>
<th>[35, 25]-light</th>
<th>[40]</th>
<th>[23]</th>
<th>[18]</th>
<th>SSE-1</th>
<th>SSE-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>hides access pattern</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>server computation</td>
<td>$O(\log^3 n)$</td>
<td>$O(\sqrt{n})$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>server storage</td>
<td>$O(n \cdot \log n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>number of rounds</td>
<td>$\log n$</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>communication</td>
<td>$O(\log^3 n)$</td>
<td>$O(\sqrt{n})$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>adaptive adversaries</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

Borrowed from:
Other SSE Solutions

<table>
<thead>
<tr>
<th>Properties</th>
<th>[35, 25]</th>
<th>[35, 25]-light</th>
<th>[40]</th>
<th>[23]</th>
<th>[18]</th>
<th>SSE-1</th>
<th>SSE-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>hides access pattern</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>server computation</td>
<td>$O(\log^3 n)$</td>
<td>$O(\sqrt{n})$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>server storage</td>
<td>$O(n \cdot \log n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>number of rounds</td>
<td>$\log n$</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>communication</td>
<td>$O(\log^3 n)$</td>
<td>$O(\sqrt{n})$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>adaptive adversaries</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

...but the search complexity is always linear in the number of matching documents

Borrowed from:
Non-Symmetric Encrypted Search

• Two variants: Practical and “not so much”

• Practical: CryptDB and Arx (CryptDB v2)
  – Deployed at scale
  – Application layer constructs

• Theoretical: Based on Oblivious RAM
  – Seek to hide everything except the result size
  – Often multi-round, *stateful*, and have heavy communication costs
Availability
Authenticated DoE

• Goal: cryptographically prove that a named resource does not exist

• DNSSEC
  – Prevent forging or modifying DNS records
  – Allow the DNS to prove that a query answer does not exist

• Current version allows for zone enumeration
DNSSEC NSEC

example.org

a.example.org

d.example.org
NSEC5: Preventing Zone Enumeration (Goldberg et al.)

- Main result: public-key operations are necessary to keep responses fresh and prevent zone enumeration
- Idea: replace the hash in NSEC3 with a keyed hash
Signed Records

Secondary keys: \( (PK_S = e, SK_S = d) \)

For each record \( x \),

\[
S(x) = (h_1(x))^d \mod N
\]

\[
F(x) = h_2(S(x))
\]

For hash functions \( h_1 \) and \( h_2 \)
Proving Non-Existence

On query q
- Compute $S(q)$ and $F(q)$
- Return $S(q)$ and hashes after $F(q)$ in order

To verify a response
1. Check that $(S(q))^e = h_1(q)$, using $PK_S$
2. Check that $F(q) = h_2(S(q))$ is before all hashes