Preventing Metadata Leakage in Communication over Public Channels

Jacob Lindahl, Masuhara Hidehiko, Youyou Cong (Tokyo Institute of Technology)

Background

With trustless technologies becoming ever-more prevalent on the modern Internet, the need arises for a new type of private messaging protocol: one that preserves the privacy of its users to the fullest extent possible without the need to trust any third party. Current implementations of privacy-focused messaging protocols suffer from a number of problems, ranging from metadata leakage to usability issues. We describe a new message transmission protocol that addresses metadata leakage and broadcast efficiency.

A portion of this protocol involves proxy servers sending anonymized messages on behalf of users. To prevent messages from being altered by rogue proxies without compromising the identities of those users, we use a zero-knowledge proving system.

We explore the implications of implementing zero-knowledge proofs in two different programming languages: Circom, a ZKP-specific DSL, and Rust, a general-purpose programming language.

Findings

Abstractions

Circuits are compiled to a rank-1 constraint system (R1CS), which is a branchless arithmetic circuit. This severely limits the kinds of abstractions available to the higher-level language.

Rust – Circuit generation must be independent of input; all abstractions are available.

Circom – Templates are the only abstraction.

```
template Sha256_2() {
    // ...
}
```

-and-

```
component hasher = Sha256_2();
```

Constraint Generation

Constraints take the form of $(\vec{s} \cdot a) \times (\vec{s} \cdot b) - (\vec{s} \cdot c) = 0$.

Rust – Graph generated by procedurally linking targets.

```
builder.connect(var1, var2);
```

Circom – Graph generated by declaratively expressing relationships.

```
hasher.a <== sequenceNumber;
hasher.b <== secretIdentifier;
```

Public Inputs

Verifier-visible parts of the witness generation.

Rust – Stream-based.

```
let sequence_number =
    builder.read::<Variable>()
```

Circom – Functional composition, marked at top level.

```
signal input hash;
```

-and-

```
component main { public [hash] } =
    Circuit();
```

Private Inputs

Witness-hiding, inputs are not visible to the verifier.

Rust – Hinting: new I/O at proof generation.

```
let output_stream =
    builder.hint(VariableStream::new(),
        HashHint::new());
```

Circom – All inputs (signals) are private by default.

Proposal

We use the **Rust** implementation of the Plonky2 zero-knowledge proof system to implement a circuit for the proxy server. The circuit proves that a sequence hash is generated with a sequence number and secret, and that that secret is used to encrypt the message payload, but without revealing either the sequence number or secret. This will require the use of a hash gadget and an encryption gadget. Proof verification on the message repository will then prevent an adversarial proxy (or frontrunner) from posting a different message under a sequence hash received from a user.

Future Work

- Extend this functionality to group (n-to-n) messaging.
- Support rotating secrets à la Signal Double-Ratchet.

References

- Buterin, Vitalik. “Quadratic Arithmetic Programs: From Zero to Hero.” *Medium* (blog), December 13, 2016. https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649.
- Gabizon, Ariel, Zachary J. Williamson, and Oana Ciobotaru. “PLONK: Permutations over Lagrange-Bases for Oecumenical Noninteractive Arguments of Knowledge,” 2019. Cryptology ePrint Archive. https://eprint.iacr.org/2019/953.
- Goldwasser, S, S Micali, and C Rackoff. “The Knowledge Complexity of Interactive Proof-Systems.” In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, 291–304. STOC ’85. New York, NY, USA: Association for Computing Machinery, 1985. https://doi.org/10.1145/22145.22178.