DuckyZip:
Provably Honest Global Linking Service

Nadim Kobeissi[1]

Symbolic Software
nadim@symbolic.software

Abstract DuckyZip is a provably honest global linking service which links short memorable identifiers to arbitrarily large payloads (URLs, text, documents, archives, etc.) without being able to undetectably provide different payloads for the same short identifier to different parties. DuckyZip uses a combination of Verifiable Random Function (VRF)-based zero knowledge proofs and a smart contract in order to provide strong security guarantees: despite the transparency of the smart contract log, observers cannot feasibly create a mapping of all short identifiers to payloads that is faster than $\mathcal{O}(n)$ classical enumeration.

Keywords: privacy-enhancing technologies, verifiable random functions, smart contracts

1 Introduction

As the internet landscape continues to evolve, so does the demand for efficient and secure web tools. One of these widely used tools is the URL shortening service, which turns lengthy URLs, like those from Google Maps, into concise links. These services have been particularly useful for social media platforms, QR codes, and any other medium where space is a precious commodity.

URL shortening services can be considered one example of a service that is entrusted to provide a mapping between a short identifier and a payload. In the case of URL shorteners, the payload is a URL. The trust model is such that users expect that the service will always map the short identifier to the same payload. However, this is not currently guaranteed by any URL shortener or similar web service.

To give other examples, the same trust model applies to the following services:

- **Pastebin:** A service that provides a mapping between a short identifier and a text payload.
- **Keybase:** A service that provides a mapping between a short identifier and a public key payload.
- **OneDrive:** A service that provides a mapping between a URL (or credentials) and a file payload.

All of these services could potentially present different payloads to different users from the same short identifier.

To address these concerns, this paper presents DuckyZip, a novel global linking service which is provably honest without revealing knowledge of any short identifier or linked payload: despite the transparency of the smart contract log, observers cannot feasibly create a mapping of all short identifiers to payloads that is faster than $\mathcal{O}(n)$ classical enumeration, thanks to a Verifiable Random Function (VRF)-based zero knowledge proof. The following sections of this paper will provide a comprehensive examination of DuckyZip's mechanics, its use of smart contracts and VRF, and how may help establish a new security standard for global linking.

2 DuckyZip Design

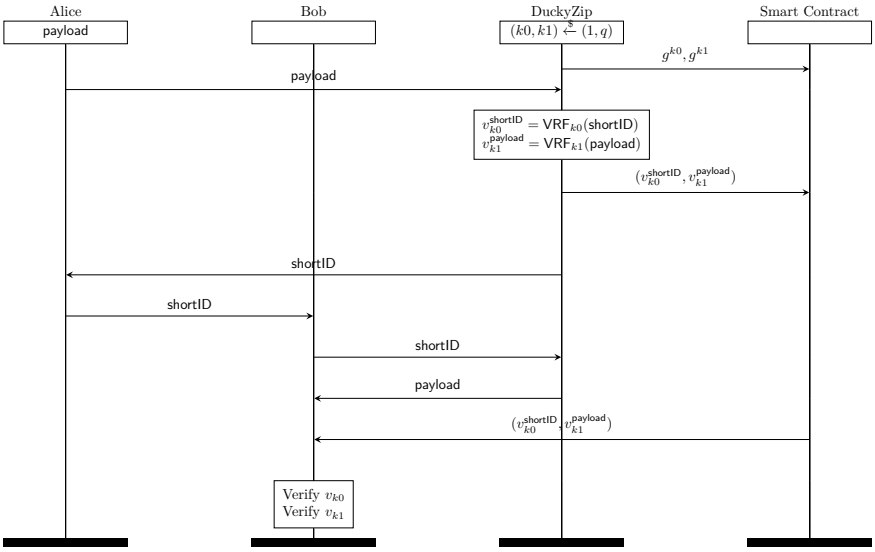

Figure 1. DuckyZip protocol description. Here, Alice is shortening a payload and sending the short identifier to Bob so that he may obtain the linked payload using DuckyZip, while also being able to verify the payload's authenticity via smart contract calls and local VRF execution.

DuckyZip depends on two cryptographic components:

– **Authenticated append-only log:** a smart contract is a self-executing contract with the terms of the agreement directly written into code that resides and operates on a blockchain network. These contracts automatically enforce

and execute the agreed-upon conditions, facilitating, verifying, and enforcing the negotiation or performance of a contract without a need for a third-party intermediary.
– **Verifiable Random Function (VRF):** a type of pseudorandom function where the owner of a private key can generate a proof to verify the authenticity of the output, given a specific input. The verifiable aspect means that any party, with the public key, the input, and the proof, can validate the output, but they can't predict it without the private key.

2.1 Designing the Distributed Append-Only Log

Certificate Transparency (CT) [1] attempts to enforce honesty with regards to SSL certificate issuance by having a number of certificate authorities and browser vendors maintain independent cryptographically authenticated append-only logs of all issued SSL certificates. This served as the initial inspiration for our design. However, since we do not possess the resources of Google or its certificate authority partners, we are unable to team up with a bunch of companies and are constrained to a smart contract on a public blockchain.

DuckyZip's smart contract contains the following data structures:

– **VRF public keys:** g^{k0} and g^{k1}.
– **Key-value store:** $\mathsf{VRF}(k0, \mathsf{shortID}) \longrightarrow \mathsf{VRF}(k1, \mathsf{BLAKE3}(\mathsf{payload}))$.

2.2 The Need for a VRF

A natural question that may arise could relate to the need for a VRF in the first place. Why not just use the smart contract to map short identifiers to payload hashes?

While this would indeed accomplish DuckyZip's main goal of honest global linking, the open nature of a smart contract's state would cause DuckyZip's entire history of short identifiers to payload mappings to become public knowledge in real time. This could violate user privacy, since it allows anyone to see all the payloads that are being submitted

Our goal, therefore, is to make it so that retrieving a list of mappings from short identifiers to arbitrarily large payloads is no less onerous in DuckyZip's case than it is in that of traditional web services: namely, the adversary would have to try all short identifiers one by one, resulting in a classical $\mathcal{O}(n)$ cost.

A naive approach towards accomplishing this could be to simply replace the mapping of $\mathsf{shortID} \rightarrow \mathsf{payload}$ with $\mathsf{H}(\mathsf{shortID}) \rightarrow \mathsf{H}(\mathsf{payload})$, where H is a secure hash function [2]. Realizing that such an approach is insufficient, one could propose to replace H with a maximally memory-hard password hashing function such as scrypt [3, 4]. The VRF approach however provides stronger still security guarantees than the latter approach, without the associated performance impact.

We use the simple VRF construction proposed by Melara, Blankstein, Bonneau, Felten and Freedman [5], which we summarize here: for a group $\mathscr{G}$ with a

generator g and of primer order q, the prover chooses a random $k \xleftarrow{\$} (1, q)$. The VRF also depends on two hash functions modeled as random oracles:

1. A hash function which maps to curve points $\mathsf{H}_1 : \star \to G$.
2. A hash function which maps to integers: $\mathsf{H}_2 : \star \to (1, q)$.

The VRF is then defined as $v = \mathsf{VRF}_k(m) = \mathsf{H}_1(m)^k$.
To prove the correctness of the VRF output:

1. Prover chooses $r \xleftarrow{\$} (1, q)$ and transmits the values (v, s, t) where:
   - $v = \mathsf{VRF}_k(m) = \mathsf{H}_1(m)^k$
   - $s = \mathsf{H}_2(g, h, G, v, g^r, h^r)$
   - $t = r - s \cdot k \pmod{q}$
2. Verifier then checks that $s = \mathsf{H}_2(g, h, G, v, g^t \cdot G^s, \mathsf{H}_1(m)^t \cdot v^s)$

2.3 DuckyZip's Protocol

DuckyZip's operation can be described through the simple protocol shown in Figure 1:

1. Alice sends a payload to DuckyZip.
2. DuckyZip generates a pseudorandom short identifier, commits a set of VRF proofs to the smart contract, and sends the short identifier back to Alice.
3. Alice sends the short identifier to Bob.
4. Bob accesses the short identifier and obtains the payload from DuckyZip.
5. Bob is then free to query the smart contract and to independently verify the existence and correctness of VRF proofs linked to DuckyZip's VRF keys and the short identifier/linked payload relevant to this instantiation of the protocol.

2.4 Practical Considerations

Some practical considerations relevant to the above protocol components:

- **Smart contract platform:** we deploy the smart contract on the Optimism Layer 2 Ethereum platform [6] in order to minimize gas cost.
- **Trustless smart contract querying:** On both DuckyZip's side and the client side, the relatively new Helios Ethereum light client [7] may be used in order to carry out smart contract calls without needing to trust a third-party service (such as Infura) and without needing to run a full node locally.
- **Write-once key-values:** DuckyZip's smart contract does not accept duplicate dictionary entries for the same short identifier by not accepting the same VRF outputs. This works because same-key VRF outputs are deterministic.
- **Short identifier search space:** In practice, a short identifier would be a 13-character string selected out of the set of allcase alphanumeric characters. This gives us a reasonably large search space for the short identifier strings ($(26 \cdot 2 + 10)^{13} \approx 2^{80}$) while still allowing the full short identifier to fit in a QR code without needing to increase the QR code's resolution.

3 Conclusion

This paper introduced DuckyZip, a first-of-its-kind, provably honest global linking service, offering robust security guarantees. Leveraging zero-knowledge proofs based on Verifiable Random Functions (VRFs) and Ethereum's smart contract platform, DuckyZip is designed to prevent the selective provision of different payloads for the same short identifier. Our design ensures that the mapping from short identifiers to arbitrarily large payloads is confidential and authenticated, offering a significant improvement in user privacy.

Future work could involve further optimization of the DuckyZip system, and exploring its integration into existing web infrastructure. We also envisage extending the techniques used in this paper to other internet tools that may be vulnerable to similar types of attacks.

Furthermore, cost can be an issue: even when using the Optimism Layer 2 Ethereum smart contract platform, the above approach can reach up to $0.20 USD per submission. In the future, we may investigate a Merkle tree structure to potentially optimize smart contract storage costs.

A working implementation of DuckyZip is available at `https://ducky.zip`, along with its source code.

Acknowledgements

We thank Lúcás Meier for his feedback throughout the authoring of this paper.

References

[1] Ben Laurie. "Certificate transparency." In: *Communications of the ACM* 57.10 (2014), pp. 40–46.

[2] Jack O'Connor et al. *BLAKE3: one function, fast everywhere*. 2021. URL: `https://blake3.io` (visited on 07/14/2023).

[3] Colin Percival and Simon Josefsson. *The scrypt password-based key derivation function*. Tech. rep. 2016.

[4] Joël Alwen et al. "Scrypt is maximally memory-hard." In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2017, pp. 33–62.

[5] Marcela S Melara et al. "CONIKS: Bringing key transparency to end users." In: *24th USENIX Security Symposium (USENIX Security 15)*. 2015, pp. 383–398.

[6] OP Labs. *Optimism: Ethereum, Scaled*. 2023. URL: `https://www.optimism.io` (visited on 07/14/2023).

[7] A16z. *Helios: A fast, secure, and portable light client for Ethereum*. 2023. URL: `https://github.com/a16z/helios` (visited on 07/14/2023).