Reclaim Protocol : Privacy preserving consensus to export reputation from webservers

Adhiraj Singh, Madhavan Malolan, Abhilash Inumella

December 30, 2022

Abstract

Webservers are not censorship resistant. They can arbitrarily delete users from their database. The user has no recourse. It is impossible for them to later prove that they indeed were a user on the webserver, and had accrued some reputation. Not being able to generate a proof of accrued reputation also has implications on interoperability. The user cannot carry any proof with them that they had some reputation on *webserver*1, and present that proof to *webserver*2 which can be verified without having to interact with *webserver*1. **The Reclaim protocol enables the user to export their reputation from the custody of the webserver into their self custody**. The proof, once generated, can be verified offline. This protocol relies on the **TLS protocol** to generate this proof, and requires *no change* on the webserver's end.

1 Introduction

Let's start with an example. Alice is a developer. She has been contributing to the repository `acme/monorepo` on GitHub.com. She would like to prove to her audience that this indeed is true. She has two options:

1. She can share a screenshot of Github.com home page after logging in that shows the list of repositories she has contributed to. This however, is easy to fake with some basic image editing software.

2. She can share her username and password with her audience. This is a bad idea for obvious reasons.

The Reclaim protocol enables Alice to open the Github.com and generate proof that `acme/monorepo` is one of the repositories on her profile page. This

proof generated by Reclaim Protocol is tamper proof - so her audience can verify and trust that she indeed is a contributor to `acme/monorepo`.

The reclaim protocol is utilizes the TLSv1.3 protocol to generate proof of the request and response exchanged between a user and a webserver **without revealing the user's secrets** such as username, passwords or API keys. Other related work include TLSNotary and DECO.

2 Protocol Description

2.1 Step 1/3 : Proof of reputation generation, via TLS Receipts

This step generates proof that certain data was exchanged between the client and the webserver without revealing secrets from the client to any party. This step is further broken down into two parts:

- TLS receipt generation – generating a transcript of requests and responses exchanged between client & server.
- Parsing Reputation – Parsing the above transcript to extract the reputation.

2.1.1 TLS Receipt Generation

This step involves the following parties:

- Client – the party wishing to prove their reputation on a webserver
- Verifier – mediates the interaction between the client & server, generates a signed *TLS receipt* and parses reputation
- Server – the webserver from where the reputation needs to be exported. This is usually a website or API provider

It works as follows:

1. Client C requests Verifier V to connect to webserver W at host H and port P
2. Verifier V accepts C's request and establishes a connection to Webserver W, and acts as a proxy server between C and W.
3. V is now privy to the *encrypted* requests and response exchanged between C & W

4. Client C establishes a TLSv1.3 session to W via V. C performs the handshake and begins to send request data meant for webserver

5. C's request can be expressed as a list of cipher-blocks

$$< Req_1^{pub}, Req_1^{sec}, Req_2^{pub}, ... >$$

   - Req_i^{pub} are the public request blocks, they contain information necessary for V to validate later that the correct request was made to W.
   - Req_i^{sec} are the secret blocks which contain private information like auth tokens or api keys - this information is never revealed to V.

   C has control over what data is sent in which block. It encrypts each of the blocks separately. This ensures the public and secret blocks all have separate `IV`s and keys for decryption and can be selectively revealed to V later.

6. W responds to C's request and sends back a response denoted by blocks Res_i^{pub}.

7. C sends all the keys and IVs to V that are required to decrypt the blocks Req_i^{pub} and Res_i^{pub}. As shown in figure **??**.

8. V generates a transcript of the request and response blocks exchanged between C and W.

9. V signs $(W, transcript, timestamp)$. This is the TLS receipt generated by V.

Security Considerations

1. MITM

   - **Certificate verification:** Verify that the certificate the server has sent, was issued by a trusted Certificate Authority.
   - **PSK:** the client can establish a TLS connection without the verifier, and use the session ticket sent by server to connect again, but this time through the verifier. As resumed TLS sessions require knowledge of a PSK (pre-shared key) — the verifier, not having the same, cannot MITM the connection.

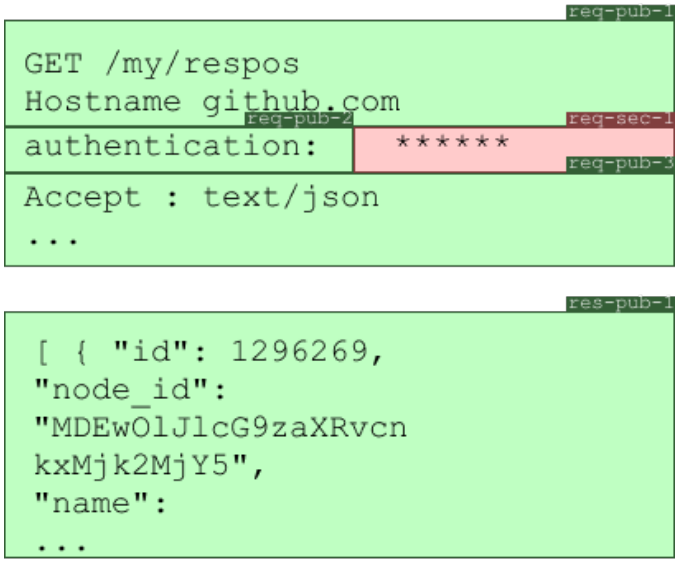

Figure 1: Public and private cipher blocks in a request and response, masking the private information

2. Computing IV - IV for $block_i$ can be computed if the verifier knows the IV for $block_{i-1}$.

   (a) Given the client wants to transmit information like $Req^{pub}1$, Req_1^{sec}, Req_2^{pub} — the following TLS messages must be sent

      i. `Encrypt(`Req_1^{pub}`)`
      ii. `KeyUpdate()`
      iii. `Encrypt(`Req_1^{sec}`)`
      iv. `KeyUpdate()`
      v. `Encrypt(`Req_2^{pub}`)`
      vi. `...`

   The `KeyUpdate()` can be performed out of band, without the involvement of V.

2.1.2 Parsing Reputation

Reputation is parsed as

$$Rep = Parse_W(transcript)$$

where $Parse_W$ is a function that parses the transcript to extract the reputation from webserver W. The verifier signs $(W, Rep, timestamp, I)$, where Rep is an encoding of reputation the client is trying to claim & I is the unique ID of the transcript.

2.2 Making the proof generation trustless, via a smart contract

The above TLS Receipt generation assumes that V is trusted. We will now relax this assumption.

The user initiates a request by sending a transaction to the smart contract on the blockchain. The transaction includes information like

- webserver W
- reputation Rep to be parsed
- payment to generate the proof

The smart contract uses a pseudo-randomization function with the above parameters to generate a list of Verifiers $<V_1, V_2, \ldots V_N>$ and θ for each request to generate proof of reputation, where N is the number of verifiers assigned to each request and θ is the minimum number of verifier's signatures needed to mint the proof of reputation.

The Client must execute the proof of reputation generation protocol with each of the verifiers in this list and store the proofs signed by the verifiers for minting the proof of reputation on chain, as we'll see in the next section.

2.3 Making the proof interoperable, by minting it on-chain

Once the User has conducted the proof of reputation generation ceremony with all the verifiers, the user would have the signatures from a subset of these verifiers. The user will then share these signed attestation to the smart contract in the form of an onchain transaction.

The smart contract will verify if the signatures are provided by the verifiers that were present in the list of verifiers generated in the previous step. If more than θ of the verifiers provided the signature, the smart contract will *mint* the proof of reputation on chain. This onchain minted proof of reputation can be queried any time and verified without the involvement of the Verifiers or Webserver.

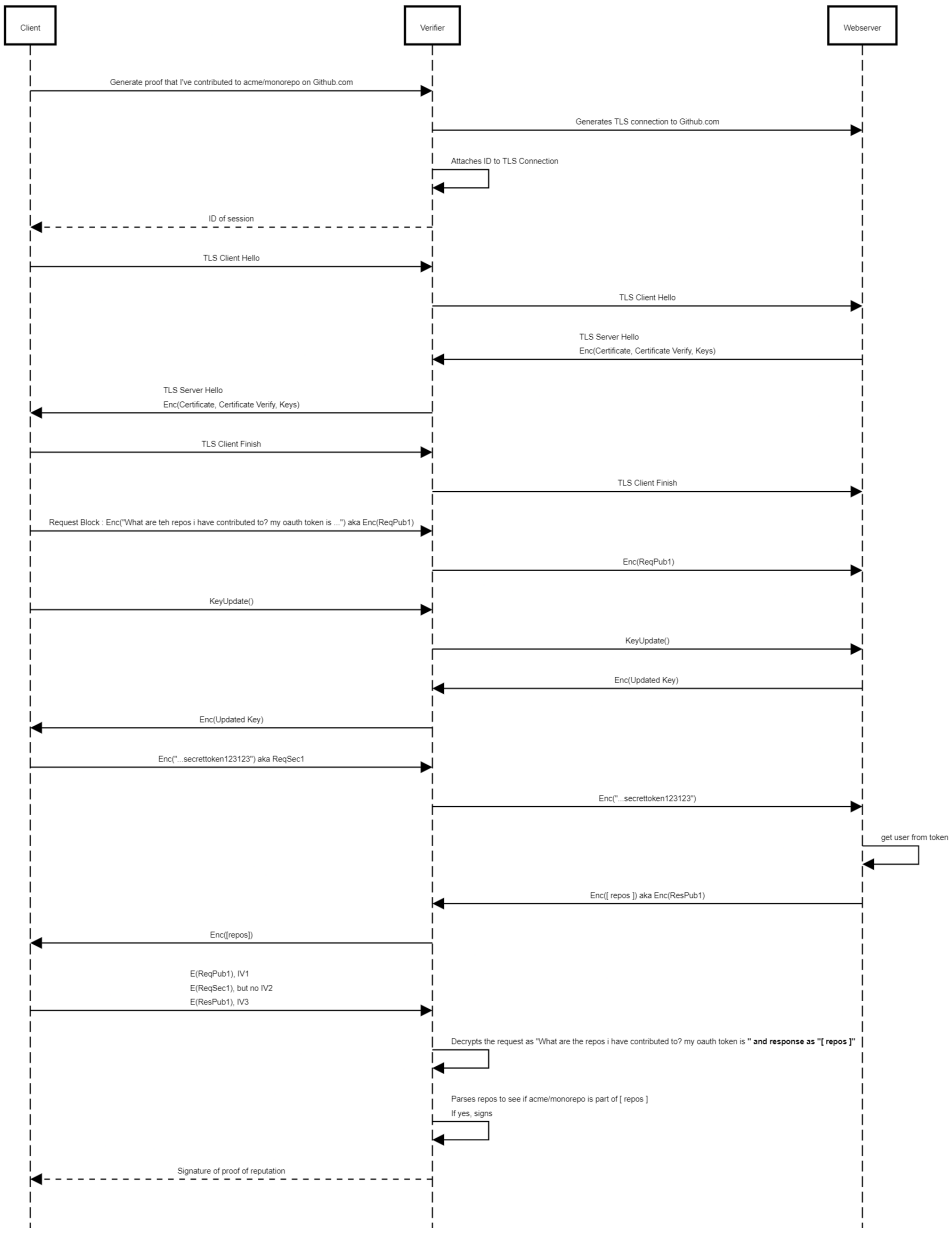

Figure 2: Illustrated example for proof generation on Github.com

3 TODO@madhavan : Economics