EID - Final Report
Estonian e-Residency-based KYC for Ethereum Smart Contracts

Introduction

The core value proposition of blockchain protocols like Ethereum and Bitcoin is in creating permissionless value transfer network, where money becomes de facto a public, shared good. But can permissionless blockchain protocols revolutionize more than "just" money?
The Ethereum protocol was invented with the specific task to answer this question, thanks to its generalized state and its ability to run complex smart contracts.
One of the hardest, open problem of modern society, for which Ethereum could facilitate a solution, is to how to evolve the concept of personal identity from the current national government sanctioned model to a digital-first, borderless and trustless one.

Without a common, open-sourced oriented effort, there is the risk that private efforts launched by Internet behemoths like Facebook and Google, would take the lead, with serious concerns for privacy and equality of access.

Attempts to establish a digital personal identity protocol without relying on external trusted authorities have been already made in the past, the most important and long-lasting being the Open Pretty Good Privacy standard (OpenPGP).
In the PGP model, each person generates a personal asymmetric key pair. This key can be used by the sender to sign documents, emails or messages to preserve their integrity during communication and enabling the recipient to attest their authenticity. The sender's key can also be used to encrypt those documents to the recipient public key to maintain their confidentiality.
Public keys can be exchanged either through a physical meeting with the other party or thanks to the Web of Trust(WoT). People can take part and extend the WoT, by signing the keys of trusted, known peers. Thanks to these links, it is theoretically possible to authenticate a person without even physically meeting them.

Unfortunately this model has a number of drawbacks: secure key management is difficult, the user experience is complex and the WoT is not well connected, making it difficult to authenticate persons without physically meeting them. Moreover, key loss or theft is disastrous because the user's personal WoT build over time is completely reset. For these reasons, today OpenPGP is only used by a reduced number of people, with extensive technical expertise and high-security demands, in limited sized groups.

In recent times, there has been a renewed interest in the identity problem, both online and offline. On one side, there has been a higher concern for the privacy of communications due to extensive governmental surveillance programs and by the rise of social networks. On the other side, the ongoing migration crisis has brought challenges for both humanitarian and security agencies trying to identify migrants where previous data is either lacking or it cannot be trusted. A number of projects and initiatives have re-explored the topic in recent years[1]. By surveying what these initiatives have produced so far, three lines of development can be identified:

- The concept of self-sovereign identity based on cryptographic keys is carried from the PGP model, but augmented by smart contracts/smart signatures schemes for better security and flexibility.
- Social-based revocation and re-issuing schemes of personal identity keys are introduced, reducing one of the main drawbacks of PGP (e.g key-loss and key-theft).
- Claims regarding personal informations should be separated from identity. For example, the date of birth, place of residence, place of work all constitute a claim made by a trusted actor: the hospital; the billing company; the employer. E.g a passport is a governmental-backed claim regarding a number of personal information.

The approach presented in this project is compatible with these recent developments in self-sovereign identity research. It is in fact a claim, made by a national government, that the asymmetric, cryptographic key pair residing on the smart card is controlled by the person matching the card's associated information.

The eID Project

The eID project leverages the Estonian e-Residency - a Public Key Infrastructure (PKI) program started by the Estonian Government that enables anyone, regardless of one's nationality or residence, to get a smart card, called Digi-ID, containing digital certificates attesting the cardholder's personal information. The verification process is done by the Estonian government and it requires the physical presence of the applicants with their own government-issued IDs in a number of approved Estonian Embassies around the world.

These smart cards have a secure element chip which stores two digital certificates, one used solely for authentication purposes and another that can be used to sign or encrypt documents with high-security guarantees.
What makes the eResidency smart-cards special among the government-issued PKI smart cards is the openness of Estonian government towards non-governmental and original uses: the smart cards are based on a well-known and widely used PKI standard and different tools are available to developers to integrate the eResidency program in their projects.

[1] http://www.weboftrust.info/

The eID project leverages the eResidency smart-card, a system of smart contracts and the Oraclize infrastructure to enable:

- Linking of any Ethereum addresses to the person's serial number, which it is tied to the personal information instead of being tied to the certificate stored in the physical smart card.
- Controlling of transfers from a smart contract-based Wallet, which can receive and send Ether or any ERC-20 based tokens, between other cardholders or Ethereum addresses, with nothing else than their smart card.

The eID project also fits right in place with Oraclize's core philosophy of bringing data to smart contracts without being a trusted party. The data here is incorporated of the signatures and the public key certificate, which are sent to the smart contract for verification.

The verification requires three elements to be checked:

- The RSA signature - signed by the holder's private key held within the smart card, on the Ethereum address the user wants to link to, or in the case of transfers, a combination of variables pertinent to the secure sending mechanism.
- The Digi-ID's certificate - needs to be validated against the certificate authority of the eID.
- The revocation list - this is a list of revoked (due to theft or lost of the Digi-ID) digital identity cards hosted by the Estonian Government.

The design implemented in the code accompanying this document is somewhat different from the one proposed in the grant application. Initially, the smart contract system was suppose to verify the RSA signatures natively during smart contract execution. Unfortunately, the protocol upgrade enabling this feature[2] has been postponed and it is still undetermined as to when it will be enabled. For this reason, the current iteration of the project delegates the verification of RSA signatures to an off-chain secure environment via Oraclize's "computation datasource"[3].

Smart Contracts System Architecture

The smart contract system implemented is composed of a number of modular parts working in unison, with the goal of providing a future-proof and extensible system:

- *Connector*: the connector enables each smart contract of the architecture to interact with each other, and if necessary, to upgrade them without disruption to the service for users.
- *Interface*: the entry point of the smart contract system, which can be used by users directly or by web UIs integrating the system

[2] https://github.com/ethereum/EIPs/pull/198
[3] https://blog.oraclize.it/overcoming-blockchain-limitations-bd50a4cfb233

- *Database*: keeps a store of all the information required for validation of the EIDs along with their most recent validity status.
- *Logic*: responsible for extracting the pertinent information from certificates on-chain, that is to be used for validation.
- *IdOracle*: a bridge contract, specific to the linking of EIDs to addresses, which connects to datasources required for the verification of the certificates, such as the off-chain computation source and an API which checks the revocation status of the certificates and validates the returned results internally.
- *WalletOracle*: a bridge contract, specific to the wallet portion, which has a similar functionality to the other oracle contract, but makes use of additional parameters, for validation of transfers.
- *WalletContainer*: accounts for any balances held by or send to EIDs, and then allows the holders of those EIDs, to interact with the contract or accompanying UI and transfer portions of their balances to other cardholders or ethereum addresses.

Verification Steps in Detail

There are a number of steps entailed in the verification process, with a slight variance in some of the steps required for linking an EID to an Ethereum address and using wallet portion, which will be denoted separately. For reference, any points with a sublist heading of a. is specific to the EID address linking and sublist heading b. is specific to the EID wallet and transfers.

1. The holder's digital signature certificate is extracted.

2. a. The extracted certificate is used for signing the hash of a message, made up of the ethereum address that will be linked with the EID in the smart contracts.

2. b. The certificate signs a message hash, consisting of the recipient's information which will be either an Ethereum address or an EID serial number. Additionally, the message includes a parameter for the origin address of ERC-20 tokens, the amount of tokens to send, a readable note to be included, and a sequential nonce to avoid replay attacks. For simplicity's sake, ether will be referred to as a token, and in the case of ether, the origin token address specified will be 0x000... or the null address.

3. The preceding two steps are done in an off-chain context, and may be done so either through the convenience of a provided UI, or for the security conscious, these exact steps may be replicated by themselves with their tool of choice. The resulting parameters must now be sent on-chain to the appropriate contract.

4. a. The interface contract will need to be accessed via the link EID function, and the parameters to be provided are:

- Extracted certificate in hex bytes
- Corresponding signature of the sender's address in hex bytes

4. b. The wallet container contract will need to be accessed, with a number of functions being potential entry points. The potential ones are:
- transferEtherToAddress
- transferTokenToAddress
- transferEtherToSerial
- transferTokenToSerial

In this case, an ether transfer to an Ethereum address will be assumed. The parameters required for the function are:
- Extracted certificate in hex bytes
- Signature of encompassing transfer parameters, which follow, in hex bytes
- The 20 bytes Ethereum address of the receiver
- The integer of ether base unit (wei) value to send
- The string form note to include
- The integer form nonce

Since the function is specific to ether, it internally handles setting the token address for this transfer, to the null address, as was used during the signing procedure.

5. Upon calling one of the above functions, the certificate's information is now parsed on-chain through the logic contract, and parameters used for verification are stored in the database and tied to the sender's address, in the case of an EID link.

6. The initial verification step entails ensuring the certificate sent along, is indeed an authentic Estonian Digital Id, whose included signature was signed by an authenticated parent certificate. The list of parent certificates will be whitelisted in the database. When the corresponding parent certificate is found in the database contract, the system checks with which signature algorithm the certificate is signed with. In the case of a SHA-1 signature, there is an additional step which takes place, in calculating the hash of the certificate's body in an off-chain context, via Oraclize's computation query, as SHA-1 is not supported natively by precompiles, like SHA-256 is. In the case of it being signed with SHA-256, the body hash is calculated on-chain and immediately saved in the database, alongside the rest of the certificate's information.

7. With the body hash available to be checked, the next step is checking the certificate's authenticity, by validating the calculated hash against the RSA signature of the parent certificate, for the holder's certificate. This step is done off-chain, using the computation query. The returned result is the decoded signature. The last appropriate bytes, which in the case of SHA-1 are the last 20 bytes and in the case of SHA-256 are the last 32 bytes of the result from the computation query are checked, and ensured to match the body hash which has been stored in the database. If this step fails, the certificate failed the authenticity check, and no further steps are taken. If the signed hash matches with what

was calculated, the certificate is considered authentic at this point, but not fully validated yet, as it still needs to be checked whether this certificate may have been revoked or whether the signed message was indeed signed by the providing user, as anyone can provide the certificate.

8. a. The next parameter to be checked is the provided RSA signature of the address to be linked. Oraclize's computation query is leveraged in this case again, and the certificate's modulus and exponent sent alongside the signature. Upon callback to the on-chain contracts, the result should be the SHA-256 message digest of the sender's address. Therefore the sender's address is hashed, and checked for equality with the resultant hash. If they match, the signature is considered valid, and the linking EID's validity level is increased one step further. If the equality condition is not met, further validation stops, as it indicates the user providing the certificate, is not the owner of the certificate, as they have failed to provide a valid matching signature of the address they are sending from.

8. b. In the case of the wallet portion, the Online Certificate Status Protocol (OCSP) is queried for a response on whether the provided holder's certificate has been revoked. This portion needs to be accessed off-chain, and it is done so by use of an Oraclize HTTPS API, which queries the OCSP responder for the status. Once the callback is returned on-chain, the result is run through the logic contract for parsing, and internal status checked to ensure that the certificate doesn't have any negative status. Additionally, the provided OCSP responder certificate is checked for authenticity from the associated OCSP authenticity certificate, hardcoded into the Oracle contracts. This additional check occurs in either a case where the OCSP check passes, or in the case it indicates the certificate is revoked and card may have been stolen. This occurs, to ensure the validity and authenticity of the OCSP result itself, for maximal user protection.

9. a. In this step for the EID linking, the OCSP check referred to above is now initiated. The steps outlined are the same, and has the same application if the OCSP responder indicates the certificate is revoked or valid.

10. The resultant signature provided in the initial OCSP query result is checked to be originating from an authentic OCSP responder, through another RSA verification done off-chain using the computation query.

11. a. If the OCSP signature check passes, and the OCSP query indicated the certificate is not revoked, the EID is successfully linked to the address, and it's considered to be fully validated. The holder or any third-party may request updates of the OCSP check at any point, in case the card is believed to be compromised at some future date. On the other hand, if the OCSP signature check passes, but the OCSP query status showed it was revoked, it can be affirmed that this is indeed a stolen card and the certificate is compromised and blacklisted with a revoked flag in the database. If the OCSP signature

check has failed, then any results from the OCSP result are discarded, and as mentioned previously, additional OCSP checks may be requested to complete the validation process.

11. b. The steps outlined in 11. a. apply similarly for the wallet, except its success indicates whether or not to continue with validation of the requested transfer of funds.

- OCSP query pass + OCSP query signature check pass = proceed
- OCSP query pass + OCSP query signature check fail = cancel transfer
- OCSP query fail + OCSP query signature check pass = revoke certificate
- OCSP query fail + OCSP query signature check fail = cancel transfer

12. b. The final step for the transfer, after the provided certificate has been fully verified and added to the database, is to verify the provided signature of the transaction's parameters, to ensure it is in fact the cardholder owner, creating this transaction. The previous steps for the wallet transfer, only apply if it is the certificate's first time interact with the EID contract system. If they had previous interactions, it will skip straight to this step, just verifying the signature. The RSA signature is verified off-chain.

13. b. Once a callback is returned, containing the resultant hash is checked against the initial parameters which were provided for the transfer. In the case of an ether to address transfer, the following transfer parameters are hased on-chain to be checked:

- Receiver's ethereum address
- Token address for the transfer (null address in the case of ether)
- Amount in base units of the token being transferred
- Note to be included
- Transfer nonce

The resultant hash is matched with the expected hash from the transfer parameters. Additionally, the nonce is ensured to exactly match whatever the current nonce stored in the database is. Anytime a valid transaction is processed, the nonce gets incremented, to protect against any potential replay attacks, as even transfers with all other parameters being duplicates, will have a varying hash thanks to the nonce increment. If both of these checks pass, the requested transfer is considered valid, and forwarded back to the wallet container contract to be executed and settled.

Future Improvements

- Leveraging RSA verification on-chain, once the appropriate EIPs are passed and implemented. This would have the effect of increasing the trustless nature of the system, as the only remaining step needing to be delegated to an off-chain context, is the OCSP responder query, to check for revocation status. It should also have the effect decreasing the costs involved with running and deploying the associated smart contract system and simplify its current elaborate nature, which exists due to several off-chain processes.

- Natively returning dynamic types between contracts. Currently a workaround had to be invented, as this is normally not possible. It entails writing to a temporary pipe variable of other contracts, to pass along dynamic types. The inclusion of this feature would greatly save gas as well, as writing to storage in the EVM is amongst the most expensive actions.
- Research on how to improve privacy through either extensive use of trusted hardware or more advanced cryptographic primitives that may become present in Ethereum's Metropolis release.
- Support of additional government-based cards, which is possible even in the current architecture, assuming the same smart card and RSA setup, but different parent certificates. Additional government certificates can be easily whitelisted.

Conclusion

The eID project enables users in possession of an Estonian Digi-ID smart card to link any Ethereum address to their eResidency identity, represented by their personal serial number which can be read on the card. The status of an address can be then queried by any other smart contract on the chain directly to the system of smart contracts.
Users can also interact with a specially designed Wallet contract to receive and send funds. The authorization for moving funds has to signed by the smart card, but the possession of funds is linked to the personal code of their eResidency identity. This resolves a major problem of dealing with cryptocurrencies based asset for non-expert users, namely key loss and secure-key handling, although this comes at the cost of trusting the Estonian government.
Key theft is avoid because the smart card acts as a trusted computing environment from which the key cannot be extracted.
Key loss can be circumvented because, if the smart card is lost, a new smart card can be issued to the same eResidency identity. After repeating the verification process for the new smart card, the Wallet smart contract can give the access to the funds back to the user.