Truffle Payroll

This document will go through the decisions taken and some of the research conducted when developing the Dapp.

The Idea

The developed Dapp is meant to be a payroll system with basic functionality implemented on Ethereum. The key benefit of implementing a payroll system on the blockchain is to:reduce the risk for employees when it comes to getting paid for their work. With traditional payroll systems, employees are paid on a monthly basis without any strong guarantee that the employer will pay in full and on time. These systems are characterised by 2 limitations that Truffle Payroll aims to address:

1. With traditional payroll system, employees are forced to trust that the employer will be willing and able pay their salary as agreed in the employment contract.

2. Employees give value to their employer on a daily basis, however, they are paid only paid after a whole month of work. Ideally, employees are paid constantly as they give value to their employers. The frequency on payment would also reduce the risk to which employees are exposed in terms of not getting paid.

Overcoming these limitations was considered as the requirement and evaluation criteria for the developed Dapp.

The idea was partly inspired by Andreas M. Antonopoulos's keynote on *money streaming*[1].
As a starting pointing when implementing this Dapp, existing systems and sample code on payment channels in Solidity were referenced:

- https://github.com/Xuefeng-Zhu/payroll
- https://github.com/guisantos/Payroll-in-Solidity

[1] https://www.youtube.com/watch?v=gF_ZQ_eijPs

- https://github.com/NFhbar/Ethereum-Payroll
- https://medium.com/@matthewdif/ethereum-payment-channel-in-50-lines-of-code-a94fad2704bc
- https://github.com/ethereum/solidity/blob/develop/docs/examples/micropayment.rst

Parts of the code were taken as a starting point, analyzed and adapted to build the desired Dapp.

Design of the Dapp

Truffle Payroll was split into 3 contracts:

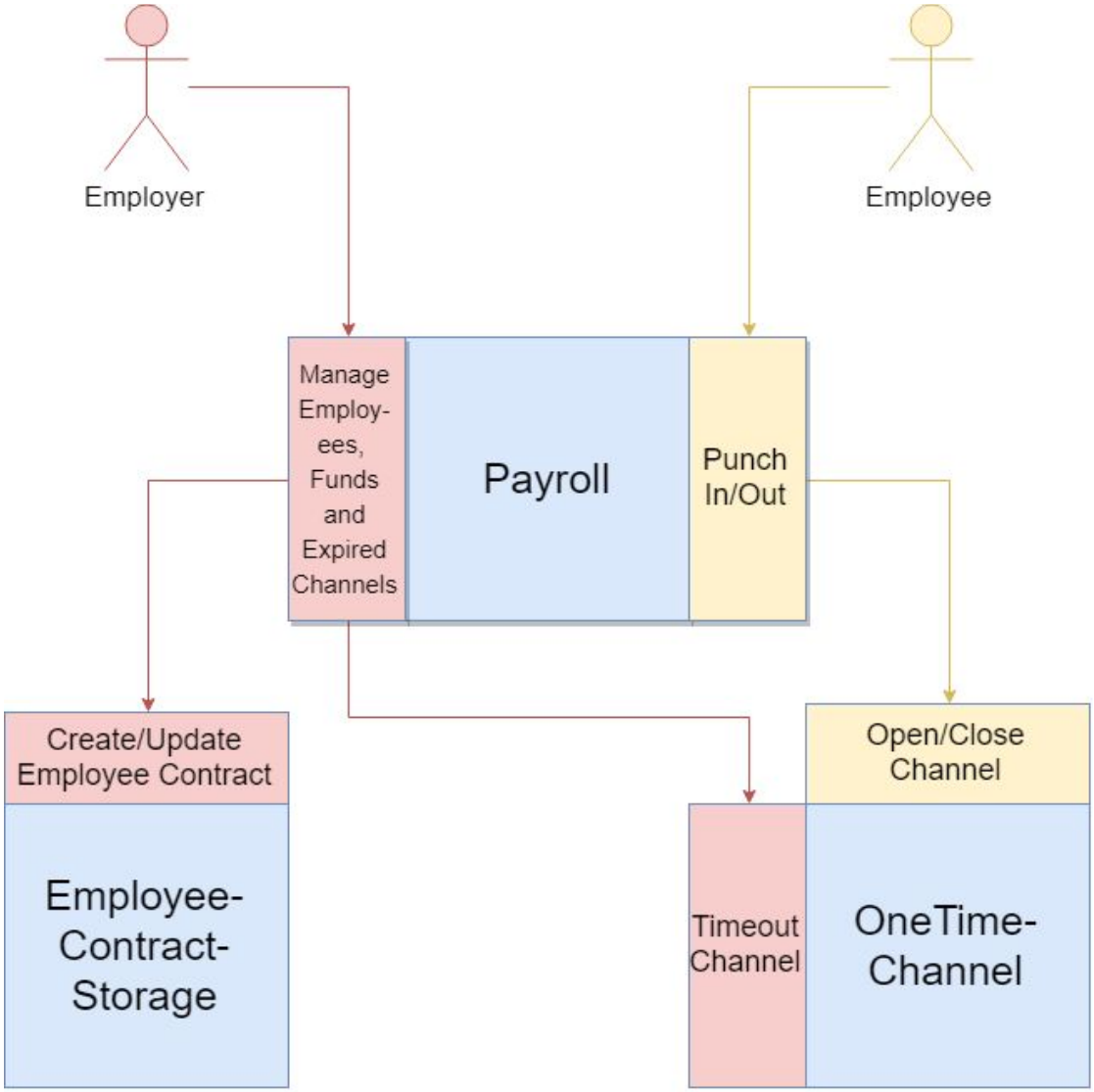
1. The Payroll contract - Contains key logic and access control related to the payroll.
2. The EmployeeContractStorage contract - Used to store all data related to employee contracts.
3. The OneTimeChannel contract - Used to verify hash, signature and value data upon punchout, paying the employee if successful. Its lifetime is managed by the Payroll contract, created on each punch in and destroyed on each punch out.

The EmployeeContractStorage was designed as a separate contract to store all data related to employee contracts. This was deemed beneficial as storing, managing and migrating data on the blockchain is expensive. By extracting it into a separate contract, access controlled by a whitelist, it is possible to reused and preserve this data in case the Payroll logic changes. New Dapps can also make use of the data if authorized.

Having a payment channel was also seen as a separate concern, more general than a payroll system. Being a separate contract also ensures that the channel logic is kept simple and limited to one employee session, thus reducing the risk of ending up in an unexpected state and locking a large sum of funds. While this decision has a high cost, due to the gas required to deploy a new contract on every punch in, it was deemed a fair tradeoff.

High-Level Design Diagram

Below is a high-level design diagram of the developed Dapp. Access control ensures that all interactions follow the designed flow. In typical usage, Employers and Employees call the Payroll contract which in turn calls the EmployeeContractStorage contract and OneTimeChannel contract.



Implementation

This section will discuss some key aspects of this Dapp. While security and design patterns were already addressed in other supporting documents, the implementation of this Dapp also dealt with complexities when it comes to dealing with payment channels and time. The following subsections are dedicated to outline some of the key decisions taken during development based on the research conducted.

Payment Channels

Based on the conducted research, a key decision when dealing with payment channels was to either:

- Have a single channel between both parties persisted over multiple transactions, in this case work sessions, or
- Have a payment channel live for the scope of a single work session.

The former approach has the advantage of being the cheaper option as the channel contract is only deployed once for each employee, regardless of the number of punchins. However, when generating hashes for such channels, the use of a nonce is required to ensure that pair of signature and hash are only claimed once for a single payment. Without the nonce, a single pair of information could be used by an employee multiple times.

The latter approach has a higher gas cost but is simpler to implement and manage. Also, the latter approach has the advantage that it is less risky in terms of funds stored in the channel and complexity. For these reasons it was the chosen approach for this Dapp.

The hash used to sign the message has 2 components:

1. The payment channel address
2. The value of the payment

The 1st component is required to ensure that the hash is only used to claim funds from the specific payment channel it is intended to. Given that a different channel is opened for each

employee, and only that employee can interact with it, this component of the hash prevents employees from ‘stealing’ another employee’s hash to claim funds from their own channel.

The 2nd component used to make up the hash is the payment value. This component ensures that an employee only claims the specific amount funds decided by the employer (based on the calculations of salary and session duration).

Apart from the hash, a signature is required to claim a payment upon punch out. This signature ensure that the hash was created by the rightful owner of the payroll. Without a signature, employees can create hashes themselves and use them to claim funds.

The channel contract verifies this information before issuing any funds:

```
// get signer from signature
address signer = getSignerFromHashAndSig(_hash, _sig);

// signature is invalid, throw
require(signer == channelOpener, "Message can only be signer by the channel opener.");

// was "proof = sha3(this, value);"
bytes32 proof = keccak256(abi.encodePacked(this, value));

// signature is valid but doesn't match the data provided      You, 25 days ago • Remove sample contracts. Update channel. Setu
require(proof == _hash, "Signature was correct but the value being withdraw does not match that specified by the signer.");
```

Lastly, a risk of payment channels is that the payee holds the payer’s funds ‘hostage’ and refuses to close the channel. To mitigate this risk, a timeout function was introduced through which the Payroll owner can expire the channel and claim back his funds.

A future improvement on this implementation could be to implement and extendExpiration() function for the channel. However this was deemed of lower priority and not included due to time constraints.

Time

When developing Dapps on Ethereum, it is not possible to get an exact value for the current time within the smart contract context. By nature, Ethereum is a distributed system which makes no particular system clock more ‘right’ than that of other nodes. The approximate time can be retrieved using ‘now’ which is an alias for the ‘block.timestamp’ value. This value represents the time at which the current block was started. Within the degrees of a few seconds, this timestamp

can be manipulated by attackers[2]. The only guarantee given by the system is that the value of 'now' cannot be less than the timestamp of the previous block[3]. In its current state, ethereum typically has a block time of less than 30 seconds. Another characteristic of Ethereum is that clients typical reject outlying timestamps as a safeguarding mechanism[4]. Albeit these measures, attackers can still manipulate the timestamp, within limits, for short periods of time as this was taken into consideration when deciding whether to get time from solidity or pass it from the front-end.

In Truffle Payroll, time was needed in different instances. Most notably:

1. Getting the punch-in time, used when calculating session duration and amount of locked funds.
2. Getting the current time used when calculating the current session duration during signing of payments. This determined the payment values.

In the 1st instance, the punch-in time is obtained using the value of 'now' from Solidity. Despite the risk of being inaccurate, obtaining the current time from the front-end would have meant that employees would be forced to trust their employer to provide the correct timestamp. This would go against the 1st identified evaluation criteria.

In the 2nd instance however, the time was obtained from the front-end. The 1st reason behind this decision is that hashes in this Dapp are designed to be generated continuously. Generating the hash based on a the value of 'now' from Solidity would require a transaction to get an accurate value[5], especially on local blockchains which have a higher block time. This would defeat the purpose of having payment channels guaranteeing money continuously with a single transaction. Retrieving it from front-end is placing some trust on the employer. However, the employee would immediately realised that the payment value is not as expected when he receives the information and could choose to punch out instantly, limiting the risk to a few seconds. As a safeguard, this value is also verified upon punch out to ensure it is not much greater than the time worked according to the 'now' of Solidity.

---

[2] https://consensys.github.io/smart-contract-best-practices/known_attacks/#timestamp-dependence
[3] https://ethereum.github.io/yellowpaper/paper.pdf
[4] https://github.com/ethereum/go-ethereum/blob/836c846812a903258f0612556481d96b3fa98758/p2p/discover/ntp.go#L46-L57
[5] https://solidity.readthedocs.io/en/v0.4.24/contracts.html#view-functions

Evaluation

In this section, Truffle Payroll is evaluated in terms of the previously states criteria namely:

1. To eliminate the need for trust by an employee that an employer will be willing and able pay their salary as agreed in the employment contract.
2. To give value back to employees as frequently as possible rather than rewarding them monthly.

Given the nature of smart contracts, Ethereum eliminates the need for trust between employees and their employer. Truffle Payroll eliminates need for trust that an employee is paid in full is eliminated by:

1. Storing the agreed salary per time unit worked on the blockchain which the employee can verify.
2. Automatically calculating the worked session's salary based on the stored employee's contract data and computed session duration.
3. Locking the maximum possible funds from the payroll upon punch in, ensuring that they are not otherwise spent by the employer.

The other area of trust required by traditional payroll system is that payments are done on time. With this Dapp, employees initiate the payments themselves rather than relying on employers. This ensures that payments happen at the agreed interval.

In order to overcome the 2nd limitation, Truffle Payroll provides a way by which employees can be paid daily according to the amount of seconds worked. On punch in, the maximum possible salary for the session is withdrawn from the employer's Payroll contract and stored in a Payment Channel contract. For every second worked, the employee is issued a hash, signature and payment value from the employer. On punch out, the employee will use the latest set of information received to claim the owed funds for the day, depending on the number of seconds worked. These can be verified by the employee to ensure their validity and that the payment value matches what was agreed. When a punch out is successful, the Payment Channel is closed and the contract destroyed, returning the rest of the funds, if any, to the Payroll contract.

Conclusion

Based on the chosen evaluation criteria, the developed Dapp does satisfy them. It provides a way through which employees can be given peace of mind when getting paid and, as an added benefit, employers also get a cost effective way to manage their payroll. However, given the narrow scope of this Dapp, there are multiple improvements and features which have yet to be developed before considering it a fully fledged payroll system. At the top of the list are:

1. Continuously automatically generate hashes, signatures and values for punched in employees.
2. Having employees sign their employment contracts.
3. Handle the firing/resignation of employees.
4. Possibly use an Oracle for more accurate time calculation.