Secure Transcript Transmission Using the Ethereum Blockchain

Mike Allan

Colorado State University
CS 556 Computer Security
Spring 2018
Dr. Indrajit Ray

Project Code: https://github.com/mlallan1307/ethereum-transcript-service
Video Demonstration: https://youtu.be/AFACljxJiXE

Fraudulent school transcripts are problem that may be solved by using the Ethereum blockchain. Employers today must be vigilant to ensure that applicant transcripts are legitimate. Hundreds of thousands of fake transcripts are purchased and used every year [1]. Short of calling the school and getting a verbal confirmation, there is no guarantee that a physical or digital transcript document is legitimate. Demonstrated here is a solution which uses the decentralized Ethereum blockchain. When transcript data is sent from a school to an employer using the blockchain, the employer can know without a doubt that the transcript came from the claimed school. The blockchain transactions are independently verifiable and immutable because thousands of independent systems are used to process transactions. This process costs less and is more secure than the current system. However, there is a problem with ensuring the long term security of the transcript data.

The Ethereum Contract

```
1  pragma solidity ^0.4.19;
2
3  contract TranscriptReq {
4    address public studentAddr;
5    address public schoolAddr;
6    address public destinationAddr;
7    bytes public   destinationKey;
8    bytes public   transcript;
9    bool public    isComplete;
10
11   // Contructor
12   function TranscriptReq(address sch_addr, address dest_addr, bytes dest_key) public {
13     require(sch_addr != address(0));
14     require(dest_addr != address(0));
15     require(dest_key.length > 0);
16
17     studentAddr = msg.sender;
18     schoolAddr = sch_addr;
19     destinationAddr = dest_addr;
20     destinationKey = dest_key;
21     transcript = "";
22     isComplete = false;
23   }
24
25   // Only the school is able to set the transcript data
26   // returns true if transcript data was set, false otherwise
27   function setTranscript(bytes transcript_data) public returns (bool) {
28     require(msg.sender == schoolAddr);
29
30     transcript = transcript_data;
31     isComplete = true;
32     return true;
33   }
34
35 }
```

Figure 1: The Ethereum contract code created for the transfer of transcripts

The keystone of this design is the Ethereum contract that was created for the purpose of transcript transactions. Figure 1 shows the contract code written in Solidity, which is the contract language for Ethereum. The data stored in the contract is shown on lines 4-9. This includes the wallet addresses of the student, school, and destination (employer). The destinationKey is the RSA public key of the employer and the “bytes” type allows it to be of any length. Transcript is also of type “bytes” and stores the transcript data. Finally, isComplete is set to true once the transcript data is set so that employers and universities can easily identity a contract as complete or not. The first method shown is the contract constructor. When the student creates this contract, using the web app, we must make sure that the contract is valid. At time of contract creation, the school address, destination address and destination RSA key must be given. If the addresses are empty or the key is empty then one of the “require” checks on lines 13-15 will fail, contract creation fails but it is still added to the blockchain as a failed contract creation because it is the Ethereum network that runs the contract code. No further interaction is possible with a failed contract.

Line 17, shown in Figure 1, has the important function of storing the contract creator as part of the contract. This is used by the university to check who is requesting their transcript. Since this data cannot be changed after creation and must be the creator’s address, we can trust that the requestor is not an imposter. The creator can also be validated by looking at the contract creation record on the blockchain but that is less convenient and more susceptible to user error. So now that the contract is created the data members are all readable by anyone but cannot be altered.

The only other function is “setTranscript” which takes the transcript data and the check at line 28 will reject any caller who is not using the specified school address. Just as the contract creator is validated, the university is also validated by the contract language. On the Ethereum blockchain you cannot claim to be someone else and use that address to send yourself money. The same principle applies here for contract creation and calls. All transactions are validated by thousands blockchain miners and must be confirmed over multiple blocks. There is no single point of failure that a malicious actor could exploit. So the university can be confident that they received a legitimate request, and the employer can be confident that only the specified university could have set the transcript data.

Data Encryption

The schools must abide by FERPA which “protects the privacy of student education records” [2]. FERPA dictates several scenarios where others are legally able to obtain transcript data; however this design was only intended to fit the use case of a student requesting their own transcript to be sent. Any information you put on the Ethereum blockchain is open for anyone to see, so posting plaintext transcript data is out of the question. With a modest plaintext transcript size of 750 bytes, a large RSA key is needed in order to perform encryption. A 4096 bit RSA key is only able to encrypt about 500 bytes and the transcript data could be larger. Since the public key is given by the employer, and the transcript data is known only to the university, we could easily encounter a scenario where the employer’s key was not long enough. To remedy this, I used a combination of AES and RSA encryption.

When the university transcript server determines that they need to respond with the transcript data of a student, they first generate a pseudorandom 256 bit value for the AES key and a 48 bit value for the AES counter. The aes-js library is used for AES encryption and decryption [3]. The maximum supported key size is 256 bits, and the maximum counter size is Number.MAX_SAFE_INTEGER (a JavaScript method) which on my machine was about 48 bits but this is checked at runtime on the server. To generate pseudorandom values the node crypto.randomBytes method was used. The documentation does not make a claim that this generates cryptographically secure pseudorandom numbers, but it says that “The method will not return until there is sufficient entropy available” [4]. Since a more efficient Node library to generate pseudorandom numbers could not be found, this seems to be the best option.

Using the generated AES key and counter, the transcript data is encrypted using AES counter mode. Counter mode was chosen over Cipher-Block Chaining (CBC) because in this use case I believe it works better. CBC requires that the data have a length that is a multiple of 32 bytes so padding would be required and every byte adds to the transaction cost. Counter mode reveals the exact length of the encrypted data, but the length of the transcript data is not of critical importance. Since a new key and counter are generated for every transcript request, AES counter mode is considered. So the university server performs AES encryption of the transcript data, and then it uses the employer’s public key to encrypt the AES key and counter used. The encrypted transcript data and the encrypted AES keys are sent together on the Ethereum contract call. The employer is able to use their private key to decrypt the AES key and counter, and then use those to perform AES decryption on the transcript data. The Ethereum contract does not care what is put into the transcript data field so it is the responsibility of the university to ensure that they are performing this encryption in a safe and secure manner.

System Usage walkthrough

The initial setup of the system begins with all three parties (school, student/applicant, and employer) creating an Ethereum wallet. With an Ethereum wallet the parties are identified by their wallet addresses. The employer must also generate an RSA public/private key pair that uses a 4096 bit key or greater. The university should have a database that contains the transcript information and wallet addresses of their students. Additionally, the university must setup a server that looks for, and responds to, Ethereum contracts created by students.

Usage of the system starts when the employer requires transcript verification. They give the applicant their public key and wallet address. The student uses a web app to enter the university's wallet address and the employer's public key and wallet address. The web app uses this information to create a new contract on the Ethereum blockchain. The contract code is publically available and is referenced by the web app when creating the contract. The student can use an application like MetaMask to approve the contract creation before it is submitted. The student can configure the "gas price" which determines if the transaction will be completed slow and cheap or fast and pricy. Publicly available websites can show the current Ethereum network state to help you determine the appropriate price [5]. On average, a "gas price" of 1 "Gwei" will result in the first transaction confirmation in less than 1 minute. The cost is also based on the transaction size and a contract creation here should cost about $1.00. The student's role is now complete and the transcript request contract is on the blockchain.

Next, the school runs a server that checks the Ethereum blockchain for new transactions. This server also has access to the contract code and can validate the legitimacy of a contract on the blockchain. The server checks that the university address specified in the contract is their own, and they validate that a contract was created by a known student by comparing the creator's wallet address to known student addresses in their database. If the student is found then the transcript data from their database is encrypted using the discussed combination of AES and RSA. The school makes a contract call to set the transcript data (which also includes the RSA encrypted key) on the Ethereum contract. The contract validates that this call came from the university address that was specified at contract creation. This contract call is similar in size to the initial contract creation, and follows the some "gas price" rules, so it should be confirmed in under a minute and cost about $1.00. The blockchain contract now contains the transcript data and is marked as complete.

The employer can use a web app or server to see that the contract has been completed on the blockchain. The web app validates the contract's legitimacy using the publically available contract code. The web app also checks that the destination address specified is their own and read the encrypted transcript data from the contract. The university address can also be validated as a known school by the employer. The employer knows that only the owner of the specified school address could have set the transcript data, but they must also check that this address is associated with the expected school, as school addresses are public knowledge. The web app can then use the employer's private key to decrypt the AES keys, and then use the AES keys to decrypt the transcript data. The employer now has the transcript which is guaranteed to be legitimate, the school is able to handle these transcript requests automatically, and the student's transcript data is never exposed to anyone but the intended recipient.

Responsibilities of Users

The Ethereum contract acts as a vehicle to this process, but the systems that interact with the contract have their own responsibilities to ensure the contract is used correctly and safely. The student will be associating their wallet address with this transaction. They may not want to reuse the same address that they used to send their transcript to their current employer or they may wonder why their employee is making a transcript request destined for another company. An Ethereum wallet can generate many addresses so they can easily use a different one if they wish, but must ensure that the school knows about the address they want to use. The student should also ensure that the employers public key is sufficiently long (e.g. 4096 bits or more), or they risk their transcript data being exposed.

The university is also responsible for ensuring that the RSA public key is long enough or they may violate FERPA by allowing the transcript data to be easily exposed. They must also manage a database of valid student addresses, a process for validating new addresses to associate with a student, and a process to revoke addresses if a student reports their wallet as stolen. The university is also responsible for the transcript encryption process and if they wish to do it differently than specified here then the decryption method must be disclosed to the employers.

The employer is responsible for ensuring that the student/applicant specified the wallet address of a known university or an applicant could just run their own university server that completes the transcript request contract. The university and employer are responsible for ensuring that the contract code used to create the contract is valid. A student could modify the contract code in a malicious way and create a contract using that code. A "golden copy" of the compiled contract code can be used to validate that the student did not modify the contract. According to the Ethereum documentation "Since the bytecode of the resulting contract contains the metadata hash, any change to the metadata will result in a change of the bytecode. Furthermore, since the metadata includes a hash of all the sources used, a single whitespace change in any of the source codes will result in a different metadata, and subsequently a different bytecode" [6]. There is currently no known way to modify a contract without altering the bytecode. The bytecode is part of the contract creation transaction, so anyone can validate that the bytecode used at contract creation is exactly the same as the "golden copy" provided by an independent organization.

Problems and Future Work

Using a combination of AES and RSA encryption means that the data is only as secure as the weakest link. A strong AES with a 512 bit RSA key means that the data is easily able to be decrypted because using a 512 bit RSA key is not secure. Enforcing a strong RSA key is the responsibility of the transcript server and student web app. In my design I check that the length of the public key is not less than 799 bytes. Using the OpenSSL library on Ubuntu this was consistently the size of the public key when generating an RSA 4096 bit public/private key pair. A key size of 4096 was used in the prototype because today that is generally considered safe. I was not able to find a way to reliably determine the RSA key size with only the public key. To ensure that the RSA key is secure the university must know its length. An improvement to the system would be to find a way to reliably determine the employers RSA key length is greater than 4096 bits.

Another possible improvement is to improve the security of the transcript data by removing it when it is no longer needed. One could add a contract call to remove the transcript data, but the previous contract creation/call data is on the blockchain forever. When getting a contract instance from the blockchain you are given the latest version of that contract and the contract itself does not keep a record of prior states. However, an attacker could search for the erase call and set their state to the previous block. There are about 600k transactions per day on the Ethereum blockchain so they this may take some time [7]. This would just add an additional hurdle for someone seeking transcript data. The contract could be modified to restrict the "get transcript data" call, but according to the documentation "Everything you use in a smart contract is publically visible, even local variables and state variables marked private" [8].

Regardless of what is done, the transcript data will always be on the blockchain and there is no way to remove it. That is potentially the biggest issue with this system. If a university ever accidentally published an unencrypted or under-encrypted transcript, they could never take it down. Using 4096 bit RSA is currently considered safe for an application such as email encryption that is being sent from one email server to another as someone would have to be capturing the packets over the network to get this data. It would not be a good idea to take your RSA encrypted email and post it on Facebook or Twitter though. That data will be accessible to anyone who wants it in 10 years (or 50 years) when it is likely that 4096 bit RSA will be able to be broken easily. Sure we could use 16384 bit RSA or larger but the same principle still applies and quantum computing makes any RSA key length nearly obsolete. Maybe we can say that if someone gets access to your transcript in 100 years that is fine, though I doubt this is covered in FERPA. The best option for encryption that will be secure in the long term may be a quantum resistant algorithm. However, there is no way to guarantee that any new algorithm will be unbreakable for 100 years. Algorithms thought to be unbreakable are regularly broken. We simply cannot assume that any form of encryption used on the transcript data will be secure 100 years from now. Therefore this system, in its current state, does pose a risk to the security of transcript data.

Blockchain technology is continuously improving as needs arise, and one of the co-founders of Ethereum has discussed possible options to address the issue of privacy. Privacy on the blockchain usually comes in the form of obfuscating the sender/receiver of a monetary transaction but it is also possible to modify the Ethereum blockchain so that smart contract data is obfuscated. In a Jan 2016 blog post by Vitalik Buterin, a co-founder of Ethereum, he discusses privacy on the blockchain and various ways that it can be achieved [9]. He says that there ways to address privacy issues in various use cases, but no one solution that works for them all. There is not a current estimate on when the Ethereum team may address these privacy issues. If they do not add some way to make contract data private, then perhaps another blockchain will add this ability in the future.

Another option may be to have the university set the contract data to just the AES keys (encrypted with the employer's public key) and send the AES encrypted transcript data via a secure (non-blockchain) medium. The employer could use the key from the blockchain to decrypt the transcript data. If the key works, then they know the transcript data came from the real university. With this method, the encryption would only need to be unbreakable during the period between the university contract call and the employer data decryption. The randomly generated AES keys (RSA encrypted) can be on the blockchain forever and that does not pose an issue unless an attacker is also able to intercept the encrypted transcript data transfer.

Conclusion

Using an Ethereum contract for transcript requests has its advantages and disadvantages over the traditional paper method. The contract itself can give confidence to the involved parties that they are all legitimate. The student and the university can be securely identified by their wallet address and the data can only be set by the university. The parties involved also have a responsibility to ensure that the contract used was not modified by checking the bytecode. The employer must also check that the university address specified corresponds to the correct university. This implementation is faster, cheaper, and better at preventing transcript fraud than the traditional method. The main problem with this implementation is that the transcript data, even if encrypted, may be easily exposed in the long term. We can have a reasonable expectation of data security for the next decade, but that transcript data remains on the blockchain indefinitely. The best alternative is to do the key transfer on the blockchain but the transcript data transfer off the blockchain. This system used transcript data, but with few modifications this could be used for other applications that need to ensure that remote parties are not imposters. Presently, this works best with data that does not need to be encrypted. Blockchain technology and solidity contracts offer us new capabilities for data/wealth transfer and party confirmation, but its distributed and public nature does pose a problem for data security.

References

[1] https://www.nafsa.org/_/File/_/ie_janfeb11_fraud.pdf
[2] https://www2.ed.gov/policy/gen/guid/fpco/ferpa/index.html
[3] https://github.com/ricmoo/aes-js
[4] https://nodejs.org/api/crypto.html#crypto_crypto_randombytes_size_callback
[5] https://ethgasstation.info/
[6] http://solidity.readthedocs.io/en/v0.4.21/metadata.html
[7] https://etherscan.io/chart/tx
[8] http://solidity.readthedocs.io/en/v0.4.21/security-considerations.html
[9] https://blog.ethereum.org/2016/01/15/privacy-on-the-blockchain/