DEcryption Contract ENforcement Tool (DECENT): A Practical Alternative to Government Decryption Backdoors

Peter Linder

plinder@assured.enterprises
peterlinder@alum.mit.edu

---- March 1st, 2016 --- V1.3 --- Draft for Peer Review -----

Abstract

A cryptographic contract and enforcement technology would guarantee release of a data decryption key to an authorized party if and only if predetermined contract requirements are satisfied. Threshold secret sharing can be used to eliminate the need for access to the hidden key under normal circumstances. It can also eliminate the liability and burden normally carried by device manufacturers or service providers when they store the decryption keys of their customers. Blockchain technology provides a mechanism for a public audit trail of the creation and release of the hidden key. The use of peer-to-peer mix-net network overlay technology can be added to insure that the blockchain audit trail documents the release of the key even if an all-powerful entity forces actors to act under duress.

1 Introduction

The use of encryption both for data storage and for end-to-end communications has grown dramatically due to increased levels of cybercrime and awareness of government surveillance. The benefits created by the application of cryptography to prevent crime and to protect the anonymity of individuals from oppressive governments is well understood and supported. But concerns by law enforcement agencies about their inability to decrypt the data and communications of suspected terrorists and other criminals has resulted in repeated calls by politicians for government decryption backdoors to give law enforcement agencies the access they desire. Knowledgeable security experts invariably point out that security-by-obscurity never holds up to the test of time. The same backdoors created for law enforcement could be discovered and used by criminals domestically and by oppressive governments elsewhere in the world. Moreover, domestic electronics manufacturers would be greatly harmed if foreign governments believed that domestically manufactured equipment included built-in security weaknesses.

It is also true that technically sophisticated criminals will always have the capability of adding extra layers of encryption to protect themselves. The mathematics of cryptography has been widely published and can never be made secret again.

Nonetheless, it can be argued that most users of technology are not technically sophisticated, that they primarily use the capabilities provided by default on their appliances, and that law enforcement agencies should be granted access to decryption capabilities in the performance of their duties when certain requirements are satisfied. Numerous device manufacturers and service providers have publicly argued not only that backdoors are a dangerous idea, but also that they do not want to warehouse decryption keys for all of their deployed devices. It is burdensome to be in the legal position of having to decide when law enforcement's requests for keys meet sufficient criteria, and it is extremely risky to store the keys of customers which may some day be stolen by hackers. Encryption solutions for which only the device owners possess the keys eliminates these burdens from the suppliers, but it creates its own difficulties. When consumers forget their passwords, their service providers can provide no assistance to recover the data. Devices issued by corporations and government agencies to their employees are similarly impenetrable, even when there is evidence of criminal employee behavior that warrants investigation. The use of biometrics such as fingerprint readers has become common as a way to minimize the problem of forgotten passwords, but this is far from ideal. Biometrics are user-IDs, not passwords. They cannot be changed if stolen and the same fingerprint which unlocks the phone is also all over the phone when viewed under the correct lighting conditions.

Controversy over governmental requests for decryption capabilities is not a new phenomenon. The Clipper chip project promoted by the US National Security Agency in the 1990s raised public awareness of the issue and prompted a group of the world's most respected experts to publish their concerns in 1997 [1]. Increased public debate in response to recent news stories and law suits prompted a similar publication by a group of experts in 2015 [2]. Certainly the concerns and questions raised in those publications must be addressed in any public policy debate on this topic. The 1997 publication summarized the four primary capabilities requested by the government, which included the need for decryption without notice to or consent of the user, ubiquitous international capability, the need for a rapid decryption response time of less than two hours, and coverage for communications as well as for data at rest. The 2015 publication detailed why such an comprehensive set of requirements is even more impractical now than it was in 1997, and drew attention to the fact that recent governmental requests have lacked a precise definition of the desired decryption capabilities. Setting aside such an exhaustive list of governmental desires, it is worth pondering what a more reasonable and more limited solution might look like. If decryption key releases were permitted to be visible to the user, if decryption was limited to data at rest, and if response time was undefined, then the discussion would certainly change in some important ways. Advocates of absolute privacy rights would continue to feel threatened, and governmental authorities would continue to desire more comprehensive powers. But the history of the internet is built around compromises and incremental improvements. If we all required guaranteed data delivery and provable security, then UDP would not exist and we would all encrypt using one time pads. Fortunately "good enough" solutions do have utility in reality, and sophisticated users will always have the ability to add additional layers of encryption. The role of this paper is simply to ask the simpler question and to explore implementation opportunities.

What is needed is a cryptographic key management system which protects the privacy rights of individuals and minimizes the burden on device manufacturers and service providers, yet also provides a means for law enforcement agencies to obtain access to decryption keys when appropriate predetermined legal contractual requirements have been met. Those contractual requirements should support any standard language such as "if and only if a valid federal warrant has been issued", for example. Threshold key sharing allows for a cryptographic key to be split into three keys, any two of which can be used to perform a cryptographic operation. One of these keys can be securely stored in the device and also be routinely available from the service provider, but it is incapable of being used without the second secret key known only to the device owner. A third key would be issued to an escrow agent who would hide the key, but it could be retrieved and used together with the first key under certain contractual conditions. This paper proposes to enhance this simple scheme by using blockchain technology to tie release of the third key to a cryptographic contract. Specifically, the escrow agent can use his key together with the desired legal contract and a nonce to create a new secret which he publishes on a public blockchain as a transaction for an account number which is the same as the identification number of the user's device. He can then destroy his key. At a later date, if the contract terms are satisfied, the escrow agent can publish the full legal contract, including the nonce, which provides information that anyone can use to obtain the desired decryption key from the data in the blockchain transaction. In a further proposed enhancement, the escrow agent passes an additional secret onto a fourth party called the proxy agent whose only subsequent source of information is the public blockchain. Using this strategy, the escrow agent would become incapable of accessing or re-creating his key, but could publish information on the blockchain which the proxy agent could use to re-create the hidden key and publish it. This strategy forces publication of the full contract on the blockchain as a prerequisite to regeneration of the decryption key, thereby creating an audit trail even if the escrow agent is forced to act under duress.

This paper presents a proposed implementation as outlined above. There are doubtless many ways in which specific details could be altered or improved upon. In those cases, the current details are chosen to aid in clarity with no intention of excluding other variations from consideration. It it also important to point that in order to further clarify the key concepts, this paper omits some encryption details which are required for a robust secure implementation.

2 Split Key Generation

The concept of "threshold secret sharing", also called a "(k, n) threshold scheme.", has been well known since it was invented independently by Adi Shamir [3] and George Blakely [4] in 1979 and expanded by subsequent cryptographers. In concept the secret which is split can be arbitrary data or a key which is used to encrypt other data. The key in question could be a symmetric key or the private key of a public/private key pair. In addition to creating a secret key which is broken up into n pieces, and which requires access to k of the key pieces (where k <= n) in order to perform cryptographic operations, a good key splitting algorithm must meet additional requirements. It is important that having knowledge of less than k keys tells nothing about the complete key and does not reduce security even if k-1 pieces are known. Cryptographers have proposed numerous additional features and algorithms to provide those features. One particularly attractive concept is the generation of key segments in a way which does not require the existence of a trusted "dealer". That is. the key generation may take place using a protocol during which the involved parties all exchange information with the result that each party ends up with a verifiable valid segment of a private decryption key without anyone learning the entire key, but a public encryption key is broadcast to all parties. [5][6] The selection of features and corresponding protocol are a trade-off between the needs of the application and computational complexity.

2.1 Key Generation and Storage Algorithms

There are doubtless a number of cryptographic algorithms and implementation schemes which could be used for the application discussed in this paper. For simplicity and in order to stay close to cryptography which is fairly well understood and trusted, the current proposal assumes a standard Shamir threshold sharing scheme in which the user's device is the dealer and a secure communication channel is established with the computers of the service provider and the escrow agent. Unfortunately, there are no official standards for threshold secret algorithms even though a number of products have made use of it over the years. Some relevant information is available in an expired 2010 IETF draft which is no longer valid, but is based on Shamir's proposal using Lagrange interpolation polynomials. However, instead of treating the secret as a large integer modulo a large prime number, this implementation achieves better computational efficiency and simpler encoding by treating the secret and the shares as octet strings, with each octet treated as an element of the finite field GF(256). Some previous implementations of this draft have been published [7]. Additional details of the split key generation and reconstruction algorithms will not be described here.

In this application, the split key generation would always take place when the device is first activated by the owner of the device, and could also take place again at a later date if it becomes necessary to generate a new set of keys due to compromise or loss of one of the key segments. The owner's device is assumed to contain a globally unique ID number for identification purposes, but in practice a new universally unique identification number (UUID) will be generated as part of the split key generation protocol. The device is also assumed to contain hardware and firmware support for a trusted execution environment (TEE) such as ARM TrustZone to provide attestation and secure key storage. For convenience, the device will be referred to as a smartphone, but it could be any computational device. The split key generation protocol can be described as follows:

- Step #1: On power-up, the device performs device attestation to insure that the firmware is unaltered from the expected state. If attestation fails, then the device may be allowed to continue booting, but support for built-in whole device data encryption is disabled. While some people may not consider this ideal, it does allow flexibility for the user to install other cryptographic solutions of their choice.
- Step #2: The device establishes secure connections with a service provider and an escrow agent. In the general case, the URLs for those servers would be user-entered. But in practice, the firmware on the device may have been preprogrammed with the public certificate and URL of a government-approved registrar

which provides those links, and which the trusted execution environment (TEE) protects. See note below.

- Step #3: The escrow agent generates a public/private key pair, which we will call UUID-P and UUID-S.
    - The Escrow agent keeps UUID-S (and its derivative UUID-P) secret to himself, and also generates a hash of UUID-P, which we will call H(UUID-P)
    - The Escrow agent discloses H(UUID-P) to the device (and device owner) and to the cell service provider for use as the identification number associated with the current cryptographic key set
- Step #4: The device generates a random master key Km to be used for symmetric encryption and decryption of data at rest on the device. The key Km is stored on the device securely using the TEE
- Step #5: The device executes the threshold secret sharing algorithm to split the master key Km into 3 segments. Any 2 of the 3 keys can be used to reconstruct Km.
    - The escrow agent's key Ke is securely transmitted to the escrow agent for retention.
    - The service provider's key Ks is securely transmitted to the cell service provider for retention. It is also stored on the device securely using the TEE.
    - The owner is asked to provide a passphrase. The passphrase is used to encrypt the owner's key Ko, and the result Po[Ko] is generated.
        - The owner's key Ko is stored on the device securely using the TEE.
        - Po[Ko] is stored on the device securely using the TEE.
        - Po[Ko] is made available to the device owner for recording, which will be needed to recover data from encrypted backups.
- Step #6: The device conducts a key segment verification to make sure that the transmitted keys have been successfully transmitted and received
    - The device requests the key Ks from the cell service provider and the key Ke from the escrow agent.
    - The device then verifies using the TEE that Ks and Ke can be used to recreate Km, and that Ks and Ko can also do so.
- Step #7: If the previous verification passed, then the TEE is used to encrypt all data on the device with the key Km and the appropriate decryption lock/unlock run-time routines are enabled.

Note for Step #2: In the ideal case, the owner of a device would decide on the contractual terms under which their escrow agent encryption key could be released, and they would select an escrow agent tasked with enforcing that contract. In practice, it is possible that governmental or organizational entities could decide on contractual terms which are acceptable for its members, and would hard-code into the TEE-protected firmware the certificate and URL of a registration server which would provide secure connectivity information for one or more approved escrow agents.

Note also that while a prearranged financial relationship is assumed to exist between the device owner and the service provider, there should also be a transfer of funds from the device owner to the escrow agent associated with this activation and split key generation sequence. Beyond compensating the escrow agent for their role, it will be shown how a portion of those funds are a key element of a later portion of the design. A simple implementation might include transferring bitcoin to the escrow agent as part of Step #5, but the implementation details are not important.

3 Key Usage

The simple use of threshold secret sharing in this manner can be summarized as follows:

- Data encryption and decryption is controlled by 3 keys, but any 2 out of the 3 keys can do the job.
- The device securely contains the master symmetric encryption key Km, the service provider key Ks, the owner key Ko, and the value Po[Ko] (the owner key Ko encrypted with a passphrase)
- After completion of the initialization procedure, data at rest on the device will be protected with whole-device encryption. Unlocking the device for use and access to data requires entering a code into the device, which can be a passphrase or a raw key code.
    - If a passphrase is entered, it is used by the TEE to generate Po[Ko]. If this matches the value of Po[Ko] stored in the TEE, then the device is unlocked.
    - If a raw key code is entered, then it is assumed that the entered key is Ke. Ke is then used with the locally stored key Ks to regenerate Km. If this value matches the stored value of Km, then the device is unlocked.
- If the user knows his passphrase and desires to change it, he can do so at any time.
- The service provider's key Ks is stored at the service provider, but it is not needed for routine operations.
    - The service provider assumes no significant liability or risk of key theft since that key alone cannot decrypt anything.
    - The service provider can be instructed to provide use of their key to any apparently valid law enforcement agency and also to the device owner if needed to recover data from backup in case the device becomes lost or damaged.
- The device should support generation of a new set of keys when needed. The operation can be requested by device menu, by device factory reset, or by remote command from the service provider using Ks as the authorization token.
    - This is accomplished by repeating the Steps #1-7 above.
    - Note that this is not a decryption operation but rather a change of keys for future encryption. Nor does it delete current data, but forces a new key generation sequence at the next device boot-up.
    - This can be necessary due to loss, compromise, or disclosure of the service key Ks or the escrow key Ke, or due to loss of the user passphrase (see the three exceptional situations listed next).
- Decryption of data in exceptional situations can be accomplished only by one of the following methods:
    - Encrypted data from backup can be recovered if the device is damaged or stolen by using the passphrase-encrypted owner key Po[Ko] and the service key Ks.
    - Encrypted data from the device can be recovered even if the owner forgets his passphrase by using the service key Ks and the escrow key Ke. A new set of keys can then be generated for the same device.
    - Encrypted data can be decrypted by a law enforcement agency without the owner's cooperation by using the service key Ks and the escrow key Ke.
- Each of these three exceptional situations involves disclosure of the service key Ke, the escrow key Ks, or both. In all cases the service provider should issue a remote command triggering a new key generation sequence at the next device boot-up, using Ks as the authorization token.

3.1 The Escrow Agent's Key

The existence and use of the escrow agent's key is central to the the proposals in this paper. It would be possible to end the proposal here by simply deciding to trust the escrow agent to release his key only under the agreed contractual terms. All the benefits of threshold key sharing would already be available. The escrow agent's key should be protected from theft, but again it is not capable of doing any decryption on its own. So if stolen, the generation of a new set of keys can be forced with no loss in security.

4 Creating an Audit Trail

The design described thus far accomplishes a significant reduction in the risks of cryptographic key theft or loss, and also provides a mechanism for data retrieval in cases of device loss or malfunction.

But the design proposal in this simple form still falls short of our ideals. The complete trust in the escrow agent to enforce the terms of the decryption key release contract is clearly a weakness. Even if the escrow agent attempts to act in good faith, they may be unable to do so. An organization could pressure the escrow agent with legal action to compel release of the key even though their justification falls far short of a "valid federal warrant" or whatever other contractual criteria had been previously agreed with the device owner. Even worse, violent action could be used against an escrow agent to compel action.

In theory, it would be ideal to find a way to automate the role of the escrow agent in order to guarantee their performance and to isolate them from possible duress. Integrating blockchain technology into the design provides an opportunity to increase transparency by creating an audit trail of all actions taken related to the escrow agent's key. The promise of smart contracts as described by various authors might provide additional functionality to achieve the goals. [8] But the current application contains two particular elements which smart contracts do not directly solve- the need for an oracle and the need for privacy. Knowledge of whether or not the terms of the decryption contract have been satisfied cannot be described in purely mathematical terms as a function of trustworthy facts. Some authors have proposed altcoin communities which may someday use distributed trust and proof-of-work to determine the validity of such abstract truths and create oracles for that purpose. [9] But for the time being, there is no clear method to eliminate the need for the escrow agent to function as the oracle to judge satisfaction of the contract terms. Another short-coming of smart contracts as presently available is the need to hide the key Ke or another secret which provides access to it. Smart contracts are designed to provide dependability of execution, but they offer no transactional privacy. The algorithms and internal state of smart contracts are publicly viewable by design. Researchers have proposed methods to increase anonymity. They have also introduced SNARKs and other zero-knowledge proofs into smart contract algorithms as a way to avoid revealing secrets. [10] But in all cases, items which must be kept secret must be kept so using a method outside of the blockchain.

Nevertheless, integrating blockchain technology into the design is a step in the right direction. This can be accomplished in a variety of ways. The use of various alt-coin and side-chain technologies provides a range of choices which can provide copious storage space and other flexibility. Datacoin [11], Factom [12] and Counterparty [13] in particular offer relevant solutions,. The Bitcoin community introduced the OP_RETURN operator specifically to provide support for DHT pointers into storage pools as the preferred solution strategy. For the sake of simplicity in demonstrating the key concepts, this proposal will assume that all data is stored directly onto the Bitcoin blockchain. This can be accomplished using Class-B transactions (the "multisig" method) as documented for the Mastercoin alt-coin [14]. This imposes a size limit on the amount of storage per transaction and causes blockchain bloat. The author does not endorse this method for actual production, but it suffices for conveying the concepts in this paper. It should also be noted that the transactions described in this paper are simplified for the sake of conveying the important concepts, and omit some implementation details necessary for a functional Bitcoin based application.

We assume that the escrow agent already has a private/public key pair which they use in daily business for the signing and encrypting of documents. We will call these keys Se and Pe.
Upon receipt of the escrow agent key Ke, the escrow agent can choose not to simply retain the key. Instead, the escrow agent performs the following operations:

- Step #8: The escrow agent creates an electronic document describing the contract in a simple format such as: "A data decryption contract has been issued which specifies the following requirements- ...". We will call this document doc1. The escrow agent signs this document to create Se(doc1).
- Step #9: The escrow agent creates an electronic document which states that the contract terms have been

satisfied in a format such as: "A valid federal warrant has been received for the data on the device with ID abc. Also, here is a nonce xyz". The value of H(UUID-P) from above is used in place of abc. The value xyz, which we will call nonce1 is randomly generated. We will call this document doc2. He then generates a cryptographic hash of the document, which we will call H(doc2). Note that the agent does not sign this document since it is not (yet) valid.

- Step #10: He then uses H(doc2) as the key to encrypt the escrow share key Ke which he received from the device owner. We will call this K(Ke,Hdoc2).
- Step #11: He then uses H(UUID-P) as a Bitcoin address, and makes a no-btc + fee payment to Bitcoin account H(UUID-P) with Se(doc1) stored as a Class-B transaction. Since Se(doc1) can be read by anyone using the escrow agent's public key Pe, this simply provides documentation of the existence of the contract and its terms. It also associates this contract with the data ID H(UUID-P).
- Step #12: He then uses H(UUID-P) as a Bitcoin address, and makes a no-btc + fee payment to Bitcoin account H(UUID-P) with K(Ke,Hdoc2) stored as a Class-B transaction. This effectively puts the escrow key Ke onto the blockchain in an encrypted form which no-one can decipher since they lack doc2 as well as nonce1 which it contains.
- Step #13: The escrow agent now destroys his copy of the original share key Ke. He retains only UUID-S (from which its public key and hash can be regenerated) and doc2 (which also contains nonce1).

If at a later date the terms of the decryption contract are satisfied, such as by law enforcement providing documentation to the escrow agent of a federal warrant for the data associated with ID H(UUID-P), for example, then the following operations can be performed to provide access to the key Ke=>

- Step #14: Using H(UUID-P), the escrow agent retrieves his copy of doc2, which states the terms of the contract and that they have been satisfied.
- Step #15: He then signs doc2 with his private business key Se, because the document is valid and his signature provides legal endorsement. We will call the result Se(doc2).
- Step #16: He then uses H(UUID-P) as a Bitcoin address, and makes a no-btc + fee payment to Bitcoin account H(UUID-P) with Se(doc2) stored as a Class-B transaction.
    - this puts Se(doc2) into the Bitcoin block-chain associated with blockchain account H(UUID-P) where anyone in the world can read it at any time and it cannot be tampered with.
- Step #17: Now law enforcement (or anyone else) can use the escrow agent's public business key Pe to decrypt Se(doc2) and obtain doc2.
- Step #18: They can then use the standard hash function to generate the hash H(doc2).
- Step #19: They can then use H(doc2) as the decryption key on K(Ke,Hdoc2) which is publicly readable as a Class-B transaction in the older Bitcoin transaction from Step #12 to generate Ke.
- Step #20: They can then use Ke, together with Ks from the cell service provider to decrypt the data on the device or from its backup.

The primary result of all this activity is that the existence and terms of the decryption contract have been published onto the Bitcoin blockchain where anyone can read it and it can never be altered or deleted. Moreover, assuming that the escrow agent is not compelled to act under duress, an audit trail of his actions in providing access to his share key Ke are also published onto the blockchain.

It is also worth noting that the new design contains a step in which the escrow agent generates a document stating that the terms of the contract have been satisfied, and he signs it with his private business key as legal endorsement. This is noteworthy because there is legal precedent to the concept that it is more difficult legally to compel someone to lie than it is to compel them to perform some other silent action. This is the legal basis for warrant canaries, which depend on the fact that a party can be compelled to remain silent under a gag order, but cannot legally be compelled to publish an lie. If it was possible to guarantee that all of the escrow agent's actions were executed through the blockchain, this might provide an extra layer of legal protection.

It could be said that the escrow agent can no longer be compelled to provide the share key Ke because he no longer has it, but that is disingenuous since he can provide doc2, thereby giving access to Ke. Moreover, there is no mechanism in the current design which requires that all of the escrow agent's actions are executed through the blockchain. By legal or physical means, the escrow agent could be forced to secretly provide access to doc1, and there would be no evidence that the decryption key had been compromised.

It is time to force the escrow agent to execute all his actions through the blockchain.

5 Separating the Oracle from the Secret

Providing a permanent audit trail of all actions taken related to the escrow agent's key can only be guaranteed by forcing the escrow agent to execute all actions through the blockchain. The design of the protocol must insure that any action taken by the escrow agent willingly or under duress outside the blockchain fails to provide access to the escrow agent key. As explained earlier, the current application contains two particular elements which smart contracts do not directly solve- the need for an oracle (judging satisfaction of the contract terms) and the need for privacy (hiding access to the share key Ke). The fundamental problem is that both of these functions have been assigned to the escrow agent. Achieving the desired goal requires a separation of duties. The escrow agent must function solely as an oracle who announces satisfaction of the contract terms on the blockchain at the appropriate time, but who is incapable of providing access to the share key Ke. A fourth party called the proxy agent can be introduced into the design to whom the escrow agent passes access to the key, after which the escrow agent destroys his copy of the key and also loses the ability to gain access to the proxy agent. The proxy agent, who's only source of information is the public blockchain, is tasked with re-creating the hidden key and publishing it at the appropriate time onto the blockchain, and is financially rewarded for doing so. This strategy forces publication of the decryption key release transaction onto the blockchain as a prerequisite to regeneration of that key, thereby creating an audit trail even if the escrow agent acts in bad faith or is forced to act under duress.

In order to implement this new design, the following operations should take place:

- Steps 1 to 7: Key segment generation and distribution should take place without change as indicated above.
- Step #8 (unchanged from above): The escrow agent creates an electronic document describing the contract in a simple format such as: "A data decryption contract has been issued which specifies the following requirements- ...". We will call this document doc1. The escrow agent signs this document to create Se(doc1).
- Step #9 (unchanged from above): The escrow agent creates an electronic document which states that the contract terms have been satisfied in a format such as: "A valid federal warrant has been received for the data on the device with ID abc. Also, here is a nonce xyz". The value of H(UUID-P) from above is used in place of abc. The value xyz, which we will call nonce1 is randomly generated. We will call this document doc2. He then generates a cryptographic hash of the document, which we will call H(doc2). Note that the agent does not sign this document since it is not (yet) valid.
- Step #10 (unchanged from above): He then uses H(doc2) as the key to encrypt the escrow share key Ke which he received from the device owner. We will call this K(Ke,Hdoc2).
- Step #11 (unchanged from above): He then uses H(UUID-P) as a Bitcoin address, and makes a no-btc + fee payment to Bitcoin account H(UUID-P) with Se(doc1) stored as a Class-B transaction. Since Se(doc1) can be read by anyone using the escrow agent's public key Pe, this simply provides documentation of the existence of the contract and its terms. It also associates this contract with the data ID H(UUID-P).
- Step #12: He then uses UUID-S to encrypt the value of K(Ke,Hdoc2). We will call this

K[K(Ke,Hdoc2),UUID-S]. Note that in reality, this should be implemented as a hybrid transaction utilizing a random symmetric key in order to overcome the limitation on the size of the data which UUID-S can securely encrypt.

- Step #13: He then uses H(UUID-P) as a Bitcoin address, and makes a no-btc + fee payment to Bitcoin account H(UUID-P) with K[K(Ke,Hdoc2),UUID-S] stored as a Class-B transaction.
  - this puts K[K(Ke,Hdoc2),UUID-S] into the Bitcoin block-chain associated with blockchain account H(UUID-P) where anyone in the world can read it at any time and it cannot be tampered with.
- Step #14: He then uses Hdoc2 to encrypt UUID-S in order to generate K(UUID-S,Hdoc2).
- Step #15: He then discloses H(UUID-P) and K(UUID-S,Hdoc2) to a proxy agent. The proxy agent can do nothing useful with that information at present.
  - The escrow agent may also supply his public business key Pe to the proxy agent. Although the public key should be available by other means, doing so eliminates the need for the proxy agent to make network requests for the public key which might draw attention to himself.
- Step #16: The escrow agent then destroys his copy of the original share key Ke. He also deletes his copy of UUID-S and UUID-P. He retains only H(UUID-P) and doc2 (which also contains nonce1).

If at a later date the terms of the decryption contract are satisfied, such as by law enforcement providing documentation to the escrow agent of a federal warrant for the data associated with ID H(UUID-P), for example, then the escrow agent should perform the following operations=>

- Step #17: Using H(UUID-P), the escrow agent retrieves his copy of doc2, which states the terms of the contract and that they have been satisfied.
- Step #18: He then signs doc2 with his private business key Se, because the document is valid and his signature provides legal endorsement. We will call the result Se(doc2).
- Step #19: He then uses H(UUID-P) as a Bitcoin address, and makes a no-btc + fee payment to Bitcoin account H(UUID-P) with Se(doc2) stored as a Class-B transaction.
  - This puts Se(doc2) into the Bitcoin block-chain associated with blockchain account H(UUID-P) where anyone in the world can read it at any time and it cannot be tampered with.
  - Although this gives anyone access to H(doc2) by way of the computations described above, nobody can obtain the key Ke because it has been wrapped in another layer of encryption using UUID-S, and only the proxy agent has access to the key needed to decrypt it.
- Step #20: He then uses H(UUID-P) as a Bitcoin address, and makes a one-btc + fee payment to Bitcoin account H(UUID-P). Here, one-btc is an arbitrary amount which could be any worthwhile non-zero amount.
  - This is significant because of the way Bitcoin works. Bitcoin accounts have a private key S which are used to sign transactions, a public key P which needs to be disclosed in order to accept money from another party, and a Bitcoin account number which is the hashed value of the public key, H(P).
  - What the proxy agent received earlier was the Bitcoin account number H(UUID-P) as well as an encrypted version of the key UUID-S (from which UUID-P can also be derived) needed to receive money from that account.

The proxy agent performs his duties by performing the following operations in exchange for financial compensation=>

- Step #21: The proxy agent checks the blockchain periodically. When the proxy agent sees that there are bitcoins in account H(UUID-P), which were put there in Step #20, he searches previous transactions on the blockchain into that account to find the Bitcoin transaction which was done in Step #19. Using the escrow agent's public business key Pe and the data in that transaction,the proxy agent then decrypts Se(doc2) to obtain doc2.

- Step #22: He then uses the standard hash function to generate the hash H(doc2).
- Step #23: He then uses H(doc2) to decrypt K(UUID-S,Hdoc2) which was disclosed to them by the escrow agent in Step #15 to obtain UUID-S (and its derivative UUID-P).
- Step #24: Before taking further action, he then calculates the hash value of the UUID-P calculated from Step #23 to obtain H(UUID-P), and checks it against the H(UUID-P) value which the escrow agent disclosed to him in Step #15. If they do not match, then the proxy agent goes back to Step #21.
    - This step eliminates the possibility of a bad actor transferring bitcoins into account H(UUID-P) in the hopes of encouraging the proxy agent to initiate transactions which would fail, but which nonetheless might be useful in trying to learn the proxy agent's identity.
- Step #25: If the comparison in Step #24 matches, the proxy agent can then use UUID-S and UUID-P on Bitcoin account H(UUID-P) in order to accept the one-btc. The blockchain transaction which the proxy agent uses to transfer the one-btc into his own Bitcoin account records UUID-P onto the blockchain where everyone can see it. Now everyone can know what UUID-P is.

Note that a law enforcement agency, the party who caused the decryption contract terms to be satisfied, or really anyone, can observe the blockchain to see the escrow agent's activities, and can observe the one-btc deposit into the Bitcoin account H(UUID-P). But more importantly, they can also observe the withdrawal of the bitcoins from that account. Anyone could transfer bitcoins to the account at any time. But only someone with access to UUID-S (and its derivative UUID-P) can withdraw those bitcoins and must disclose UUID-P in the process. The final steps can be executed by anyone to obtain access to the escrow agent's original key Ke=>

- Step #26: Anyone can search previous transactions on the blockchain into account H(UUID-P) to find the Bitcoin transaction which was done in Step #13 by the escrow agent and obtain K[K(Ke,Hdoc2),UUID-S].
- Step #27: Anyone can search previous transactions on the blockchain into account H(UUID-P) to find the Bitcoin transaction which was done in Step #19 by the escrow agent and obtain Se(doc2).
- Step #28: Anyone can use UUID-P from Step #25, which was done by the proxy agent, to decrypt K[K(Ke,Hdoc2),UUID-S] from Step #26 to obtain K(Ke,Hdoc2).
- Step #29: Anyone can use the escrow agent's public business key Pe to decrypt Se(doc2) from Step #27 to obtain doc2.
- Step #30: Anyone can use doc2 from Step #29 to calculate its hash value H(doc2).
- Step #31: Anyone can use H(doc2) from Step #30 together with K(Ke,Hdoc2) from Step #28 to obtain Ke, the original escrow agent's decryption share key.

In essence, the proxy agent is simply someone who waits for bitcoins to show up in an account and then takes them. But they cannot get access until the escrow agent executes the Bitcoin transaction which is the signed document proclaiming that the contract terms have been satisfied (which discloses doc2). And such a document must be issued publicly on the blockchain because unless the proxy agent sees it and discloses UUID-P to get the deposited bitcoins, nobody can decrypt K[K(Ke,Hdoc2),UUID-S] to get access to the escrow agent's original key Ke.

Note also that as promised, this design requires that the escrow agent publish a document stating that the terms of the contract have been satisfied, rather than simply requiring some silent action on their part. As stated, this is noteworthy because of the concept, also used by warrant canaries, that it is more difficult legally to compel someone to lie than it is to compel them to perform some other silent action.

6 Fully Automating the Proxy Agent

At a superficial level, it may appear that steps 8 through 31 simply transfer the burden of an escrow agent maintaining a list of secret keys Ke into the burden of a proxy agent keeping a list of secret H(UUID-P)/K(UUID-S,Hdoc2) pairs. Additionally, a proxy agent could be forced to act under duress and therefore must be protected or hidden, just as similar concerns were expressed earlier with regard to an escrow agent.

But in fact, incorporating the blockchain into the design and separating the duties of the oracle from the function of hiding the secret has resulted in a design which is very powerful. Assuming that the proxy agent can be hidden, the escrow agent is no longer in need of protection since all of their actions are clearly documented on the blockchain, thereby producing an audit trail which would expose any actions under duress. Moreover, the proxy agent has been defined in such a way that his role can be described by a very simple algorithm, and he need be trusted only to act in his own financial best interest. This makes the proxy agent very easy to protect or to hide.

As a very simple non-technical solution, the role of the proxy agent could be assigned to any person or organization geographically and politically isolated from the perceived threats. They need to be trusted only to act in their own financial best interest and to be competent to reliably execute their simple algorithmic duties.

Even more interesting is the possibility that the proxy agent could in fact be replaced entirely by an automated and protected process. The simplicity of the proxy agent's role makes this plausible.

Recent advances in distributed systems and network designs raise some interesting possibilities in pursuit of this goal. What is needed is the ability to launch a proxy agent process into a network, where it propagates to an unknown untraceable peer, waiting to transfer bitcoins to the fortunate owner of the machine it happens to reside on at some time in the future. The concept of passing an encrypted message between a series of intermediate nodes to preserve anonymity was first proposed by Chaum in 1981 [15]. Network designs built with this architecture, often referred to as Chaumian mix-nets, are similar to the well-known Tor onion routing network. Peer-to-peer versions which implement the mix-net using a network overlay without dedicated routing nodes and which perform packet delaying, reordering, and cover traffic can make this strategy particularly effective. MixMinion [16] and Tarzan [17] were early attempts at implementation. Today, Freenet [18] and I2P [19] are similar projects which are widely used and include a wide range of features including distributed file storage support. Even more recent peer-to-peer projects such as Storj [20] and MaidSafe [21] offer weaker anonymity but tie their architecture to a block chain and an e-currency in order to motivate peers to provide large amount of distributed file storage. The analysis of data durability in replicated distributed storage systems with peers of random uptime and downtime has been studied at length with the conclusion that acceptable data durability can be achieved with reasonable levels of data duplication. [20] [22] [23].

In the current context, a peer-to-peer design based on these concepts would be enhanced so that the participating peers also execute the proxy agent algorithm needed for our application.

An implementation which provides the desired functionality is currently under study. In order to provide proof of concept with reduced complexity, the first pass design of the proxy agent eliminates the possibility of adversarial peers and assumes that peers are reliable according to the following model:

- The proxy agent is set up and administered by a single organization. Funding is provided from payments made by the escrow agents upon submission of secrets, and by bitcoins which are redeemed whenever escrow contracts are satisfied and keys released.
- The proxy agent builds a matrix of virtual machines operating on three different cloud hosting providers.

Each set of VMs within a single cloud provider is called a zone.

- The proxy agent is responsible for the maintenance and financial support of the infrastructure.
- The VMs have no persistent non-volatile storage for data. The only non-volatile storage in a zone is for the operation of the hypervisor and to hold the VM boot image, which is the same for every VM within the same zone. The VM code makes each VM aware of which zone it is in.
- Each virtual machine has the following characteristics:
    - It maintains all data in volatile ram memory.
    - It runs peer-to-peer software by which it participates in a peer-to-peer anonymous mix-net overlay network consisting of all virtual machines in all zones.
    - It runs proxy agent algorithm software which supports the role of a proxy agent as described above (watching the blockchain and accepting a payment) for one or more values of H(UUID-P).
    - It maintains a short list of H(UUID-P)/K(UUID-S,Hdoc2) pairs in ram; 1/3 of the list is for H(UUID-P)/K(UUID-S,Hdoc2) pairs which it services; the other 2/3 of the list are H(UUID-P)/K(UUID-S,Hdoc2) pairs which it is holding as backup for other VMs in other zones.
    - It runs software by which it can securely exchange data with other VMs. This is used to exchange H(UUID-P)/K(UUID-S,Hdoc2) pairs with neighboring VMs, as well as to communicate some VM state information and service requests.
    - It keeps track of the number of unused locations in its service and backup lists, and communicates the number of free slots to its neighbors upon request.
- The list of hashes H(UUID-P) stored in the network is not secret. Only the H(UUID-P)/K(UUID-S,Hdoc2) pairs themselves and their locations within the network are secret.
- When an escrow agent is prepared to disclose an H(UUID-P)/K(UUID-S,Hdoc2) pair to a proxy agent, it submits it to a random peer in the proxy agent's network . That peer then passes a service request to two other random peers in the other two zones. Each of the three peers makes a request of its neighbors to inquire whether they have storage space available on their list, and the inquiries propagate until space is found. Using the returned information, the initiating peer can now construct three onion routes to the final peers, one of which will service the H(UUID-P)/K(UUID-S,Hdoc2) pair and the other two as backups. Each of the three peers is also provided with the identity of the other two peers so that they can periodically cooperatively verify that all three are alive and well, and assign new VMs if necessary. The initiating peer then deletes his copy of the secret and the routes. The layered onion route encryption insures that the secret and its locations in the network were never disclosed to anyone other than the four peers involved.
- It is impractical or impossible to examine the states of the running virtual machines to determine which H(UUID-P)/K(UUID-S,Hdoc2) pairs are being served by which virtual machines, and there is no persistent storage which could be searched.
- The data durability of the H(UUID-P)/K(UUID-S,Hdoc2) pairs is dependent entirely on the reliability of the cloud hosting providers and the fact that the hosting is three-way redundant.
- Network capacity can be checked by making an inquiry to a random peer, who can make space inquiries to his neighbors, and who can accumulate the total of the propagated responses.
- Network function and reliability can be easily tested by inputting dummy H(UUID-P)/K(UUID-S,Hdoc2) into the proxy agent network, and putting bitcoins into the appropriate accounts to see if the network responds.
- In the case of catastrophic failure, new keys can be issued to all affected devices. The list of hashes H(UUID-P) whose keys were stored in the network is known. It can be forwarded to the service providers with a request that a new key generation sequence be triggered on the next boot-up of each device on the list.

In the future, it would be desirable to extent the proxy agent implementation to a peer-to-peer network running on an affiliation of organizations in numerous countries or even on the open internet. Such an implementation would present additional challenges. The voluntary nature of peers joining and leaving the network would create a challenge in guaranteeing data durability for every K(UUID-S,Hdoc2) value. The possibility of adversarial peers raises significant complications. Since only a small percentage of H(UUID-P)/K(UUID-S,Hdoc2) pairs would ever result in bitcoin payment due to key release, a financial incentive would likely be required to motivate sufficient voluntary peers. But ultimately, most of these problems are identical to those encountered in any typical blockchain application. Thus, there is good reason to believe that the application requirements could be achieved by implementing the proxy agent peer-to-peer network as its own e-currency backed blockchain or sidechain.

7 Conclusion

We have proposed a system for the management of data decryption keys which protects individual privacy rights, minimizes the risks of key loss or data loss, and provides a mechanism for law enforcement agencies to obtain access to keys when appropriate predetermined legal contractual requirements have been met. We started with the extreme of encryption in which only a single party holds the key. We showed how threshold secret sharing provides reduced risk of key loss or data loss without sacrificing privacy in most situations. We showed how the same technology makes it possible for a key escrow agent to facilitate decryption when contractual conditions are met, even though his key alone is incapable of performing decryption. We introduced blockchain technology to provide an audit trail of creation and possible release of the escrow agent's key. We further introduced peer-to-peer mix-net network overlay technology to insure that the blockchain contains a complete audit trail even in cases when the escrow agent acts in bad faith or is forced to act under duress. This combination of technologies has been proposed here in the context of decryption key management. In the more general case, blockchain data storage combined with an untraceable process hidden in a peer-to-peer mix network also has relevance as a general method of protecting oracles in smart contract applications.

References

[1] Hal Abelson et al, “The Risks of Key Recovery, Key Escrow, and Trusted Third-Party Encryption” (1997); http://academiccommons.columbia.edu/download/fedora_content/download/ac:127128/CONTENT/paper-key-escrow.pdf

[2] Harold Abelson et al, “Keys Under Doormats- Mandating insecurity by requiring government access to all data and communications” (MIT 2016-07-06); https://dspace.mit.edu/bitstream/handle/1721.1/97690/MIT-CSAIL-TR-2015-026.pdf

[3] A. Shamir. “How to Share a Secret”; Communications of the ACM, vol. 22, no. 11, pp. 612–613, 1979; http://cs.jhu.edu/~sdoshi/crypto/papers/shamirturing.pdf

[4] G.R. Blakley, “Safeguarding Cryptographic Keys”; Proc. Am. Federation of Information Processing Soc. (AFIPS ’79), Nat’l Computer Conf., vol. 48, pp. 313-317, 1979; https://www.computer.org/csdl/proceedings/afips/1979/5087/00/50870313.pdf

[5] Torben Pryds Pedersen, “A Threshold Cryptosystem without a Trusted Party”; D.W. Davies (Ed.): Advances in Cryptology - EUROCRYPT '91, LNCS 547, pp. 522-526, 1991; http://link.springer.com/content/pdf/10.1007%2F3-540-46416-6_47.pdf

[6] Marco Carpentieri, “Some Democratic Secret Sharing Systems”; Discrete Applied Mathematics 59 (1995) 293-298, Received 1 March 1994; revised 8 August 1994; http://www.sciencedirect.com/science/article/pii/0166218X9580007Q

[7] https://github.com/antik10ud/threshold-secret-sharing and https://github.com/seb-m/tss

[8] Ethereum: https://www.ethereum.org/

[9] Hivemind: http://bitcoinhivemind.com/

[10] Ahmed Kosba, Andrew Miller, Elaine Shi, Charalampos Papamanthou, Zikai Wen, “Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts”; https://eprint.iacr.org/2015/675.pdf

[11] Datacoin: http://datacoin.info

[12] Factom: http://factom.org/

[13] Counterparty: http://counterparty.io/

[14] Storing Mastercoin data in the blockchain: http://omnichest.info/files/mscappendix_draft.pdf

[15] David L. Chaum, Univ. of California, Berkeley , “Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms”; Communications of the ACM, Volume 24 Issue 2, Feb. 1981, Pages 84-90; http://freehaven.net/anonbib/cache/chaum-mix.pdf

[16] G. Danezis, R. Dingledine, and N. Mathewson, “Mixminion: Design of a type iii anonymous remailer protocol”; Security and Privacy, 2003. Proceedings. 2003 Symposium on, pages 2–15. IEEE, 2003; http://mixminion.net/minion-design.pdf; https://github.com/mixminion/mixminion

[17] M. Freedman and R. Morris , “Tarzan: A peer-to-peer anonymizing network layer”; Proceedings of the 9th ACM conference on Computer and communications security, pages 193–206. ACM, 2002; http://freehaven.net/anonbib/cache/tarzan:ccs02.pdf; https://pdos.csail.mit.edu/tarzan/

[18] Freenet: A distributed decentralized information storage and retrieval system by Ian Clarke (1999); https://freenetproject.org/assets/papers/ddisrs.pdf; https://github.com/freenet/

[19] I2P: https://geti2p.net/en/

[20] Storj: http://storj.io

[21] MaidSafe: http://maidsafe.net

[22] Sara Alouf, Abdulhalim Dandoush, and Philippe Nain , “Performance Analysis (of data lifetime) of Peer-to-Peer Storage Systems”; http://www-sop.inria.fr/members/Philippe.Nain/PAPERS/P2P-STORAGE/ITC2007.pdf

[23] S. Ramabhadran and J. Pasquale, “Analysis of Durability of Replicated Distributed Storage Systems”; *Proc. International Parallel & Distributed Processing Symposium (IPDPS)*, Atlanta, April 2010; http://cseweb.ucsd.edu/~pasquale/Research/Papers/ipdps10.pdf