Key Recovery: Inert and Public

Colin Boyd[1], Xavier Boyen[2], Christopher Carr[1,2] and Thomas Haines[2]

[1]Norwegian University of Science and Technology, NTNU, Norway
[2]Queensland University of Technology, QUT, Australia

Abstract

We propose a public key infrastructure framework, inspired by modern distributed cryptocurrencies, that allows for tunable key escrow, where the availability of key escrow is only provided under strict conditions and enforced through cryptographic measures. We argue that any key escrow scheme designed for the global scale must be both *inert* — requiring considerable effort to recover a key — and *public* — everybody should be aware of all key recovery attempts. To this end, one of the contributions of this work is an abstract design of a proof-of-work scheme that demonstrates the ability to recover a private key for some generic public key scheme. Our framework represents a new direction for key escrow, seeking an acceptable compromise between the demands for control of cryptography on the Internet and the fundamental rights of privacy, which we seek to align by drawing parallels to the physical world.

1 Introduction

Key escrow was a popular research topic, and subject of contention, during the 1990s [2] — the era of the so-called crypto wars [12]. Recently, the crypto wars seem to have resumed, principally due to the "Snowden revelations" of global mass surveillance. Whilst security advocates and technical experts have spoken out against demands from government agencies to weaken, or even prevent, the use of cryptography, governments make an opposing case, demanding greater powers of surveillance in pursuit of terrorists and organised crime.

Pros and cons of key escrow. Allowing a third party to hold unlimited escrow over cryptographic keys comes with serious security concerns. Indeed, even if the party holding the escrow is trustworthy, confidence that

this party is completely secure against forms of malicious compromise, such as subversion or hacking may not be achievable, rendering the cryptographic keys vulnerable. Consequently, accepting key escrow amounts to placing broad and considerable trust in both the character and abilities of the party holding escrow. Concerns surrounding the level of trust are not unfounded, as shown by recent examples of security breaches, directly compromising keys used by members of the public. For example, in 2015, the SIM card manufacturer Gemalto reportedly had the keys to millions of SIM cards compromised in a breach conducted by GCHQ and supported by the NSA [11].

In a widely publicised study in 2015 [1], a group of 15 eminent cryptographers and technologists re-examined governmental access to data and communications, comparing the situation now with that in the 1990s. They concluded that law enforcement access "will open doors through which criminals and malicious nation-states can attack the very individuals law enforcement seeks to defend". They further argued that it would require unreasonable costs and lead to economic loss.

Arguably, governments have had far too much power over, and knowledge of, the communications of its average citizens. This has been recognised previously. In 2013, the United States President's Review Group on Intelligence and Communications Technologies [15] recommended that "... the US Government should:

1. fully support and not undermine efforts to create encryption standards;

2. not in any way subvert, undermine, weaken, or make vulnerable generally available commercial software; and

3. increase the use of encryption and urge US companies to do so, in order to better protect data in transit, at rest, in the cloud, and in other storage."

On the other hand, there is an opinion that reasonable controls on cryptography are desirable. The oft presented motivation is the need to reveal the communications of those involved with, or suspected to be involved in, major criminal or terrorist activities. The same President's Review Group on Intelligence and Communications Technologies [15] also stated (Recommendation 20): "... the US Government should examine the feasibility of creating software that would allow the National Security Agency and other intelligence agencies more easily to conduct targeted information acquisition rather than bulk-data collection."

Bart Preneel in his 2016 IACR Distinguished Lecture, *The Future of Cryptography* [16], asked: What about the balance? He points out that privacy has both individual and collective dimensions. While intelligence agencies have arguably gone too far in their usage of technology, they may lose out in some scenarios. Preneel challenges the community to design better solutions.

Compromise solutions. During the first crypto-wars in the 1990s, cryptographers contributed a number of compromise key escrow solutions, aimed at balancing the power of law-enforcement agencies with various constraints. One such example was *partial key escrow* [4, 5] which allows escrow agents to recover keys with moderate effort but makes the computational requirements of mass surveillance prohibitively costly. Another example was oblivious key escrow [6] in which users share their keys among a large number of unknown escrow agents who must cooperate in order to recover a key. Our proposal takes inspiration from both of these ideas.

During the current crypto-wars there has been little interest from the cryptographic community to explore compromise solutions. One exception is the cMix proposal of Chaum et al. [7], part of a practical project known as Privategrity. Although the proposed system efficiently provides high level security and privacy properties, it relies on users sharing long-term keys with a fixed set of servers so that compromise of all servers reveals all user information. Such a level of trust, even if extensively distributed, has received widespread criticism. Note that compromise of all trusted nodes, and subsequent compromise of user secrets, if it should happen, can be undetectable. Importantly, we have already seen a willingness for international cooperation to record communications, and so it is not immediately obvious that this approach would achieve the desired results.

Our aim is to explore some ideas for better compromise solutions to key escrow. We are motivated by the observation that extraordinary access to personal property in the physical world does not receive the same debate and controversy as extraordinary access in the virtual world. By examining the differences between the virtual and physical world, we are led to propose some principles for key escrow which we believe can defuse much of the controversy behind earlier key recovery proposals. Furthermore, we observe that modern techniques, specifically based on ideas from decentralised cryptocurrencies, allow these principles to be realised in practice.

CONTRIBUTION The main aim of this work is to focus the attention of the community in an area that is politically difficult, with an aim of offering responsible solutions to these problems. The technical contributions can be summarised as follows.

- New principles for the design of large scale escrow systems are proposed.
- Possible methods for implementing these principles are described.
- A generic method for creating a proof-of-work system based on a public key scheme is proposed.

2 Background

This section focuses on the building blocks of our proposal. The first two of these originate from the original crypto wars: oblivious key escrow and partial key escrow. The third element we introduce is a decentralised consensus mechanism conceived with modern cryptocurrencies.

2.1 Oblivious key escrow

The first of our three proposals described later has its roots in the notion of *oblivious key escrow* [6]. Blaze suggests a form of escrow in which keys are widely distributed amongst a large pool of escrow servers numbering thousands or millions. Key recovery can be performed by a threshold of escrow servers (perhaps numbering hundreds or thousands) following some pre-agreed policy. The key-sharing is oblivious in that the key owner does not know which subset of possible share holders were selected.

The system also allows for extraordinary access following a process which Blaze calls *angry mob cryptanalysis*: if enough servers of the system agree, they can ignore the policy and recover the secret key for other member of the scheme. The idea is that such a mob would only choose to do so if there was some mass consensus that key recovery is justified. This brings to mind the recent case where Apple and the FBI were in conflict over access to data in a convicted terrorist's iPhone [10]. With angry mob cryptanalysis, the decision in such cases could be made by wide consensus.

The principle behind Blaze's idea is that a party's secret key can be recovered when a large enough proportion of people demand it. The argument for this kind of key escrow is that in order to apply it, a vast number of people must be committed to finding a secret key that corresponds to a

published public key. To get such a large body of people on board, it would be necessary to perform this process in the public eye. The argument is that even powerful agencies would have to announce their intentions in order to recover a key in reasonable time. The advantage here is removing the ability of a rogue state to simply recover the key through: ulterior means or coercion; via a legal process; or through simply amassing the power to do so. Crucial to this design is that the parameters are chosen so that coercion of such a large population of users is infeasible.

2.2 Partial key escrow

Shamir seems to have been the first to suggest the idea to escrow only part of the private key.[1] The principle is that law enforcement agencies with considerable computational power will be able to obtain some targeted keys, but will not have the resources available to perform mass surveillance, provided enough entropy remains in the unescrowed part of the key.

Using *weaker than usual* keys is another proposal that has been suggested for key escrow, with the argument that while there is less security on an individual level, by selecting appropriate parameters, it is infeasible to extract the keys for all.

However, partial and *weaker than usual* key escrow has seen a lot of criticism, with arguments against this approach reasonably stating arguments such as the following:

- One cannot make accurate assumptions on the computational resources of powerful agencies who may wish to undermine a key escrow system using weaker keys such as the NSA/FSB/MSS/GCHQ/ASD;
- Breakthroughs in cryptanalysis are unpredictable;
- Cryptanalysis techniques may not be made public, and there is little incentive to do so, as greater reward can be garnered through the private sale of cryptanalytic breakthroughs; and
- By the power of Moore's law, over time there will be a considerable reduction in the cost of recovering a key.

2.3 Ensuring a distributed user base

One of the main achievements of Bitcoin, is enforcement of the principle first described in the original Bitcoin proposal [13], namely *one CPU, one vote*.

[1] Unpublished, but widely attributed [4, 8].

Traditionally, peer-to-peer systems could be vulnerable to what are popularly referred to as Sybil vulnerabilities [9]. Sybil vulnerabilities often arise when a single party can easily and quickly create multiple pseudonymous identities. Malicious parties can perform a Sybil style attack in a threshold key escrow scheme by creating multiple pseudonymous identities — in order that they stand a greater chance of receiving more key shares.

Employing the one CPU, one vote principle attempts to solve the Sybil problem. In this scenario, it does not matter how many pseudonyms any participant in the scheme can create, what matters instead is how much work they can produce. Thus, any party wishing to pose as two entities, must provide the work of two entities, and so on linearly in the number of users one wishes to pose as. Inevitably, while one individual is able to pretend to be multiple entities, they cannot trivially increase their computational power beyond some reasonable margin.

In a distributed environment where there are multiple machines, all attempting to outpace the others, it soon becomes hard to implement any form of Sybil attack.

3 Design Principles

We are motivated by comparison between the process of cryptographic key recovery and the process of obtaining access to physical premises. This extends an analogy that is made between allowing extraordinary access to personal encrypted communications and giving access to personal property [1]. We observe that there are at least two important differences between the physical world and the cryptographic world which are not usually considered in such analogies.

1. Obtaining access to physical premises requires significant resources. This typically includes the presence of people and the use of physical equipment, both over a significant time.

2. Instances of access are difficult to hide from public view. Often they involve forced entry with multiple actors, and a form of recorded process.

Our thesis is that these two properties are inhibitors to abuse of extraordinary access, whether committed by law enforcement agencies or by legitimate property owners. Use of significant resources makes mass abuse without a valid target impossible. Public observation and records of access

instances prevents covert abuse. Our aim, therefore, is to mimic these properties in the cryptographic world so that a more acceptable compromise can be reached. This leads us neatly to two design principles.

3.1 Inert and Public

Inert. Key recovery should cost something to those seeking to recover the key. Moreover, this cost should be measurable and increase with the number of keys to be recovered.

Public. The only viable way to recover keys should be with a publicly recorded process. Every instance of a key recovery attempt should be publicly known, and the record of key recovery instances should be infeasible to alter, hide or falsify.

As we have seen in recent years, both the properties of inertia and public accountability are almost nonexistent in online communication. With unencrypted communication, the effort required does not grow with the number of keys required to compromise. For example, consider the tapping of deep sea cables. The hard work required to intercept the communication happens once, and from then on the cost of interception is tiny for all new communications using that line. In contrast we advocate a fair cost for each communication recovery.

This example also shows how the public aspect is defeated. Tapping of communications can be achieved in a covert and undetectable fashion. At it stands, our knowledge of compromises comes mostly from whistle-blowers within the system who are often legally required to remain silent, and may have to break laws with heavy penalties in order to bring the activities to public attention.

Creating a system that is both public and inert is not easily achievable in the current Internet, but there are emerging technologies that are resolving some of the constraints. In order to describe the system we propose, we next define a space where certificates for escrowed keys can be placed.

3.2 Blockchains, Decentralised ledgers and PKI

Traditional public key infrastructure (PKI) has been shown wanting for the end user, and we are still in a situation where, for the most part, secure online communication between two parties is "off" by default. This is especially true in email, where the difficulty of both obtaining and using public keys

helps to prevent its widespread use [18, 17]. Linking an identity with a public key is still a difficult problem, due to the methods for distribution and ways to assert key ownership. The web of trust model, while innovative, presents too great a challenge to the user to be effective as a world wide tool. On the other hand, over-reliance on centralised architecture is itself a major issue.

The emergence of decentralised ledger systems (or Bitcoin like systems), on the other hand, seems to provide a natural and, most importantly, a practical way to achieve the design goals since such systems are decentralised and inherently satisfy the properties of inertia and public verifiability. Indeed, using blockchains to construct public ledgers in which to store credentials or certificates for use in public key cryptography has been previously considered [19], along with alternative proposals for certified credentials within the Bitcoin system itself [3]. Our direction complements this work, considering the further requirement for key recovery under certain circumstances, whilst still maintaining strong levels of security for the majority of the system.

We have identified the following four properties as the main challenges in achieving a practical systems satisfying our goals.

Strong Keys Generating keys that are currently strong does not prevent them from becoming weak in the (near) future. One problem is to create an escrow system that can hold up for an extended period.

Resistance to Sybil Vulnerability Are the keys vulnerable to Sybil like attacks? In a key sharing mechanism, it is important that the vulnerability threshold to a Sybil attack is suitably measured.

Public Attempted recovery of the key must be made public in order to be acceptable.

Inert All mechanisms used to recover the keys require at least some amount of effort.

To satisfy the third property, we need to devise a way to allow for the users of the system to recover the secret keys for the public keys that are posted in the system. To do this, we devise a blockchain like public key infrastructure layer for users and devices, that acts in the middle ground, as well as allowing market forces to try their best. Blockchains have been proposed as a distributed public ledger before, and there are plenty of applications to choose from [13]. All we require is that the ledger is append only, and available to all members.

A central advantage of using a distributed decentralised ledger is that certificates can be uploaded to the system in a manner that resembles distributed public key infrastructure. A user asserting a key to the system will have it verified if it follows the rules of the system. The design is decentralised and traceable, so altering and faking certificates is difficult. Unlike the more centralised server storage approach, where compromising the key server gives control of the key server and allows revoking and creation of false keys, there is a much greater resistance in a proof–of–work based ledger model.

We propose two layers of infrastructure, a top layer, where authorities such as nation state, and perhaps well audited companies, remain, which we call *trusted roots*. The second layer is the key management layer, where the signed keys are appended to, and stored on, the global record.

For example, a user may want to claim their email address, sally@uni.gov along with registering their name Sally F and other supplementary information. This certificate is appended together and signed by the relevant authority that agrees that Sally is in fact the legitimate owner of the email address. By this method, there is a global consensus on trusted roots and their intermediate authorities across all platforms.

Assumption 1 (Trusted Roots) *All parties maintain an identical list of trusted root authorities, containing the root authorities, their corresponding public key, and some auxiliary information such as description and location.*

This design is similar to traditional public key infrastructure. However, this design incorporates both PKI and the assertion of certificates for individual users. These top layer authorities are axiom authorities of the system, designed to represent governmental bodies or *trusted* corporations, without which there would be no place to start. Notably, we make no judgement on whether this is ideal, however we seek to mimic the real world as closely as possible.

Assumption 2 (Certificate ledger) *The certificate ledger is append only and available to all parties. Specifically, all participants can write and read accurately to the certificate ledger. After some known period of time t, records within the ledger cannot be removed.*

4 Our Framework

We propose a framework that uses either one of, or a combination of, the oblivious key escrow method and the secret sharing of moderately weak keys.

This leads to two alternative proposals and a third which combined the first two.

4.1 Proposal 1: Decentralised oblivious key escrow

Our first proposal is to build a key escrow scheme using a distributed smart contract system. Specifically, the policy to enable release of keys, as described by Blaze [6], is embedded in a smart contract. Release of keys can then only occur when the policy is satisfied, as guaranteed by the integrity of the blockchain. This allows building oblivious key escrow into a transaction that can act as a credential. However, this requires an oblivious key escrow protocol to be secure in a form of white box execution. Users of the system would then be able to verify that the oblivious key escrow took place correctly and if so include the transaction in the system.

Using a Turing complete language, available in modern crypto-currencies like Etherium [14], it is possible to programatically enforce a random choice of escrow servers. We note that, in the original oblivious key escrow paper [6], it is possible, without any risk of detection, for the sender to collaborate with the receivers to select only the receivers of their choice.

Proposal 1 is a method of oblivious escrow where key recovery is available according to some agreed and publicly checkable policy. The use of a distributed ledger enables the property of public accountability. However, this method does not require computational effort in order to effect key recovery.

4.2 Proposal 2: Partial Key Escrow

Our second proposal is to escrow parts of a key, and record them on a distributed ledger. The system therefore should act as a form of PKI, so that anyone can verify the correct association between a user and a public key.

We desire an efficient way to include a secret within a system such that the verifiers of the distributed ledger can quickly check that credentials are included. In order to do that, we require that users select a public key of a specified length, such that it is *short enough* to recover the secret key if a considerable effort is applied for some length of time, yet *strong enough* to prevent recovery by reasonable computational resources.

Creating a good metric for the security of a public key cryptosystem of different lengths is challenging. Therefore we introduce a feedback loop mechanism between the security of the public key scheme used for key escrow

and the proof–of–work system. This requires building an alternate proof-of-work system from a public key system in such a way that recovering the secret key for a given public key can be accurately quantified. The idea is to build a proof-of-work system that relies on the speed of finding secret keys to corresponding to registered public keys.

Quantifying strength of public keys. This approach comes with another interesting feature. Since users are rewarded for their work on the proof-of-work system, there is an incentive for them to find the best algorithms and obtain the best hardware to recover the secret keys. This will be useful for determining the long term strength of any scheme deployed in this manner. If it is valuable to find weaknesses in a specific scheme, then weaknesses may be found more readily, and the absence of weaknesses being found indicates the security of that specific public key scheme. This creates a financial incentive to find weaknesses in keys, and increases the level of scrutiny of a public key scheme, from just a few interested parties, such as those interested in covert surveillance, and ones developing algorithms and software to sell to those agencies. It increases the scope to the general public, bringing the cryptanalysis of a public key scheme out into the open.

A solution to this problem is to create a public key based proof–of–work system. This means that a certain level of computational work has to be applied to a target credential in order to retrieve the secret key. This gives us a metric on the security of a public key scheme. Say, on average, every minute a secret key is found for a public key scheme with a certain level of security, then we can feed that back into the key generation process for the credential. If the key should be secure for a greater length of time, then this the key size should be scaled with respect to size of keys actually being recovered in the system.

4.3 Proposal 3: Decentralised Oblivious Partial Key Escrow

Proposals 1 and 2 each have certain benefits, but individually fall short of solving the full problem. However, by combining both proposals, we can eliminate the drawbacks of each approach.

Our third proposal is a system where to recover the secret for a user, both effort must be made and consensus must be reached. To recover a key, such that, a large population of users must agree to release a reduced key which in turn must be the target of significant computational resources. This guarantees both:

	Partial Escrow	Oblivious Escrow	Prop. 1	Prop. 2	Prop. 3
Public	✗	✗	✓	✓	✓
Inert	✓	✓	✓	✓	✓
Future secure	✗	✓	✓	✗	✓
Sign up not required	✓	✗	✗	✓	✓[2]
Sybil resistant	✓	✗	✗	✓	✓
Traffic analysis resistant	✓	✗	✗	✓	✓

Table 1: Comparison of main properties of the three proposals

- *public accountability* since the oblivious key escrow is inherently public, and weak keys will not be publicly available; and
- *inertia* since all keys released are current in their security parameters and the oblivious key escrow can have security levels tuned for ongoing security.

This means that in the future the keys will not be trivially breakable unless one has previously mounted a Sybil attack on all keys. Therefore, it is necessary to mount the attack beforehand on all users in order to compromise the public property of the system. We summarise the proposals, and their advantages in Table 1.

5 Methods for Implementation

Here we sketch ways to implement each of the three proposals. At present we are only in a position to outline a proof of concept. Detailed designs and experimental systems will require further work.

5.1 Implementing proposal 1

Implementing oblivious key escrow as a smart contract requires working within the white box execution environment, where all execution is public. Doing any cryptography in this environment is hard since all keys used would be public. So, all values need to be encrypted by the relevant parties before submitting to the contract.

[2]There is a requirement for pre-registration for the oblivious part of the key escrow.

Implementing this first proposal can be done in solidity, the programming language for the distributed *smart contract* system Ethereum [14], using the code and protocol shown below. Note, that for simplicity, we have replaced the blind signatures and anonymous channel (as specified in the design of Blaze [6]) with the pseudo-anonymity of the block chain. It should however, be easy to reintroduce them.

```
pragma solidity ^0.4.0;
contract ObliviousKeyEscrow {

    function randomGen(uint seed, uint max) constant
        ↪ returns (uint randomNumber) {
        return(uint(sha3(block.blockhash(block.number-1),
            ↪ seed))%max);
    }  //More secure randomness is preferable

    struct Receiver {
        uint publicKey; //Preferably new
        bool joined;
        bool chosen;
    }

    address sender; //The person wanting to escrow
    mapping(address => Receiver) receivers;
    mapping(uint => address) receiverNumbers;
    uint[] shares; //People who should hold shares
    uint[] encrypted_shares; //Encrypted shares of the key
    uint numReceivers;

    /// Create a new Oblivious Key Escrow
    function ObliviousKeyEscrow() {
        sender = msg.sender;
    }

    /// Join as a potential sender
    function Send(uint8 publicKey) {
        Receiver receiver = receivers[msg.sender];
        if (receiver.joined) return;
        receiverNumbers[numReceivers] = msg.sender;
        receiver.joined = true;
        receiver.publicKey = publicKey;
        numReceivers++;
    }

    /// Choose the receivers to receive key shares
```

```
    function ChooseReceivers(uint numShares) {
        if(msg.sender != sender) throw;
        if(numReceivers < numShares) return;
        uint seed = 0;
        for(uint i = 0; i < numReceivers;i++){
            seed += receivers[receiverNumbers[i]].publicKey
                ↪ ;
        }//Calculate a seed
         for(i = 0; i < numShares;i++){
            uint randomNumber = randomGen(seed,
                ↪ numReceivers);
            seed += randomNumber;
            shares[i] = randomNumber;
        }
    }

    ///Send the key shares to the relevant parties
    function SendShares(uint[] tempEncryptedShares){
        encrypted_shares = tempEncryptedShares;
    }
}
```

A person choosing to escrow their key would put the contract on the block chain. They would then wait for senders to join by calling *Send*, after which they would call *ChooseReceivers* to securely, and publicly, choose random receivers. Once the receivers are chosen, the sender calls *SendShares* with the key shares encrypted to the relevant party's public key.

5.2 Implementing proposal 2

First, we define general public key encryption and signature schemes.

Definition 1 (Public Key Encryption Scheme) *A public key encryption scheme is made up of a tuple of probabilistic polynomial–time algorithms*

(KeyGen, Enc, Dec) such that:

$\mathsf{KeyGen}(1^\lambda)$ *is the key generation algorithm, taking security parameter* λ *as input and producing a public key, secret key key tuple* (pk, sk) *respectively.*

$\mathsf{Enc}(pk, m)$ *takes the public key* pk *and a message* m*, and outputs a ciphertext* c*.*

$\mathsf{Dec}(sk, c)$ *takes the secret key* sk *and a ciphertext* c*, and outputs a plaintext* m*.*

Definition 2 (Signature Scheme) *A signature scheme is made up of a tuple of probabilistic polynomial–time algorithms (KeyGen, Sign, Ver) such that:*

$\mathsf{KeyGen}(1^\lambda)$ *is the key generation algorithm, taking security parameter* λ *as input and producing a public key, secret key key tuple* (pk, sk) *respectively.*

$\mathsf{Sign}(sk, m)$ *takes the secret key* sk *and some message* m*, and outputs a tag* s*.*

$\mathsf{Ver}(pk, s, m)$ *takes the public key* pk*, the message* m*, and the tag* s *and returns either accept or reject.*

Providing correctness requirements, informally, for a public key scheme such that $(pk, sk) \leftarrow^r \mathsf{KeyGen}(1^\lambda)$ for some security parameter λ, then the probability that $\mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) \neq m$ is a negligible of λ. Similarly for the signature aspect, for any m, the probability that $\mathsf{Ver}(pk, \mathsf{Sign}(sk, m), m)$ returns reject is also a negligible function of λ.

Remark 1 (Public Key Scheme) *We call any overarching scheme that supports both signatures and encryption a public Key Scheme.*

Definition 3 (Proof-of-Work Adaptable) *Let* P *be a public key scheme. For every* $pk_i, pk_j \leftarrow^r \mathsf{KeyGen}(1^\lambda)$ *such that* $|pk_i| = |pk_j| = n$ *and for all adversaries, the difference in expected computational steps between recovering a secret key for* pk_i *and recovering a secret key for* pk_j *is negligible in the security parameter* λ*.*

Furthermore, for any $pk_i \leftarrow^r \mathsf{KeyGen}(1^{\lambda'})$ *and* $pk_j \leftarrow^r \mathsf{KeyGen}(1^{\lambda''})$*, where* $d = |pk_j| - |pk_i|$*, for any adversary* A *that can reliably recover a*

secret key for pk_i in t computational steps, then A can reliably recover a secret key for pk_j in $t + f(d) \geq t$ steps, for some monotonically increasing function f.

In other words, keys of equal size can be recovered in roughly the same number of steps, and that there is not some selection of weaker keys within the scheme that are easier to recover. It also ensures that the difficulty increases by a known factor on the length of the key, meaning we are able to extrapolate the security of larger keys, based on that of smaller keys.

Definition 4 (Full key space) *Let P be a public key scheme. We say that P has full key space in $[i, j]$ if for every binary string of length between and including i and j, then each public key is uniquely representable as such a string, and has a unique corresponding secret key, for some choice of security parameter λ.*

With the groundwork in place, it is possible to build a proof–of–work scheme based on a public keys scheme, providing it satisfies Definitions 1, 2, 3, and 4. The process for building this PoW system is described as follows:

1. Collect broadcast transactions (or credentials), and label them as x_i.
2. Take a unique reward value y, which is the information used to claim a reward.
3. Using a suitable hash function H, apply $c_i = H(x_i, c_{i-1}, y)$ and let pk be the first d bits of c_i, where c_{i-1} represents the previous state.
4. The challenge is to find a secret key sk corresponding to pk for the given public key scheme, such that $\mathsf{Ver}(pk, \mathsf{Sign}(sk, c_i), c_i)$ returns true.

Once such an sk is found, it can then be broadcast to claim the reward. This creates a chain-like consensus mechanism to be used as the backbone for the system. Now, when credentials (or transactions) are created, we insist that they are created using the same public key scheme for the consensus ledger mechanism, but with a higher level of security than creating the difficulty. With the properties defined for the public key scheme, we can pick a key that we know is quantifiably more challenging to recover than the amount of work on the system at a given time. Now, if a key ever needs to be recovered, the same amount of energy can be expended on the recovery of a key. However, with a large enough system it is necessary to engage with the community and have them recover the key in order to achieve recovery in a timely fashion.

There must be a way of adjusting the difficulty of the system depending on the rate at which keys are being recovered. This is possible in this framework as we can simply adjust the $pk = c[0, \ldots, d]$ for some maximum difficulty d, which can vary depending on the rate at which solutions are made available.

5.3 Implementing proposal 3

Proposal three can be implemented by combining the implementations of proposal one and two. An important aspect here is the method used for the creation of credentials, as the corresponding secret key used in the credential creation must only be recoverable if both the instance of oblivious key escrow and the instance of scheme two is satisfied. Clearly these requirements are contradictory, as the secret keys chosen in scheme two are purposefully designed to be recoverable when a concerted computational effort is applied, whereas in scheme one, they are not. To solve this, we simply include two separate keys when creating a credential, where each key is implemented under either scheme. While this is not entirely elegant, we believe the extra burden on the key generator and the extra space requirement for storing the key are sufficiently small to allow for such a solution. Notably, this choice will double the number of authentication rounds, which may detract from applicability, but we consider to be a price worth paying for a PKI system which seeks to achieve simultaneous goals such as these.

6 Conclusion

The aim of this paper is to explore future directions in key escrow. We have presented three outline solutions, constructable in the no-man's-land between complete surveillance and complete security. There remain interesting open problems in this line of enquiry. From an applications perspective, a concrete proposal for a public key scheme that matches the criteria laid out in Definitions 1, 2, 3 and 4 is needed. Utilising quantum resistant public key schemes may be necessary for long term use. While we have no candidate construction, we note that for classical cryptography, there may be scope for schemes built on the discrete logarithm problem.

Of course, making the scheme truly public and inert requires a large user base, and wide adoption. For that reason, research into scale and usability of such a system would complement this work.

Leveraging existing technology. A slightly different, but otherwise parallel approach, would be to overlay the escrow credential management from proposal 2 directly on top of the pre-existing, and popular, Bitcoin system. The major advantage of doing this is that the inherent security of the system would come packaged with it. In principle, there is nothing stopping a system that utilises Bitcoin's blockchain as the decentralised append-only ledger. This can allow anyone to announce their credential, along with a signature from a trusted axiom authority within a transaction. This is trivially possible on Bitcoin, and you could do this within a transaction to announce your public key. The open problem is how to store the secret key within the transaction, so that it could be recovered by finding a preimage of an output form the SHA-256 algorithm.

References

[1] Harold Abelson, Ross J. Anderson, Steven M. Bellovin, Josh Benaloh, Matt Blaze, Whitfield Diffie, John Gilmore, Matthew Green, Susan Landau, Peter G. Neumann, Ronald L. Rivest, Jeffrey I. Schiller, Bruce Schneier, Michael A. Specter, and Daniel J. Weitzner. Keys under doormats. *Commun. ACM*, 58(10):24–26, 2015.

[2] Harold Abelson, Ross J. Anderson, Steven M. Bellovin, Josh Benaloh, Matt Blaze, Whitfield Diffie, John Gilmore, Peter G. Neumann, Ronald L. Rivest, Jeffrey I. Schiller, and Bruce Schneier. The risks of key recovery, key escrow, and trusted third-party encryption, 1997.

[3] Giuseppe Ateniese, Antonio Faonio, Bernardo Magri, and Breno de Medeiros. Certified bitcoins. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *Applied Cryptography and Network Security*, volume 8479 of *LNCS*, pages 80–96. Springer, 2014.

[4] Mihir Bellare and Shafi Goldwasser. Verifiable partial key escrow. In Richard Graveman et al., editors, *CCS '97, Proceedings of the 4th ACM Conference on Computer and Communications Security.*, pages 78–91. ACM, 1997.

[5] Mihir Bellare and Ronald L. Rivest. Translucent cryptography - an alternative to key escrow, and its implementation via fractional oblivious transfer. *J. Cryptology*, 12(2):117–139, 1999.

[6] Matt Blaze. Oblevious key escrow. In Ross J. Anderson, editor, *Information Hiding, First International Workshop*, LNCS, pages 335–343. Springer, 1996.

[7] David Chaum, Farid Javani, Aniket Kate, Anna Krasnova, Joeri de Ruiter, and Alan T. Sherman. cMix: Anonymization by high-performance scalable mixing. *IACR Cryptology ePrint Archive*, 2016:8, 2016.

[8] Dorothy E. Denning and Dennis K. Branstad. A taxonomy for key escrow encryption systems. *Commun. ACM*, 39(3):34–40, 1996.

[9] John R. Douceur. The Sybil attack. In Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, editors, *Peer-to-Peer Systems, First International Workshop, IPTPS*, LNCS, pages 251–260. Springer, 2002.

[10] Michael Hack. The implications of Apple's battle with the FBI. *Network Security*, 2016(7):8–10, 2016.

[11] The Intercept. The great SIM heist: `https://theintercept.com/2015/02/19/great-sim-heist`, 2015.

[12] WIRED: Todd Lappin. Winning the crypto wars. `www.wired.com/1997/05/cyber-rights-10/`, 1997.

[13] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `https://bitcoin.org/bitcoin.pdf`, 2008.

[14] Ethereum Network. Ethereum: Smart contract and decentralized application platform : `https://github.com/ethereum/wiki/wiki/White-Paper`, 2016.

[15] President's Review Group on Intelligence, Communications Technologies, Richard Alan Clarke, Michael J Morell, Geoffrey R Stone, Cass R Sunstein, and Peter P Swire. Liberty and security in a changing world: Report and recommendations of the president's review group on intelligence and communications technologies. `http://www.whitehouse.gov/sites/default/files/docs/2013-12-12_rg_final_report.pdf`, 2013.

[16] Bart Preneel. IACR distinguished lecture: The future of cryptography. `http://homes.esat.kuleuven.be/~preneel/preneel_iacr_dl_vienna2016.pdf`, 2016.

[17] Scott Ruoti, Jeff Andersen, Daniel Zappala, and Kent E. Seamons. Why Johnny still, still can't encrypt: Evaluating the usability of a modern PGP client. *CoRR*, 2015.

[18] Alma Whitten and J. Doug Tygar. Why johnny can't encrypt: A usability evaluation of PGP 5.0. In G. Winfield Treese, editor, *8th USENIX*. USENIX, 1999.

[19] Duane Wilson and Giuseppe Ateniese. From pretty good to great: Enhancing PGP using bitcoin and the blockchain. In Meikang Qiu, Shouhuai Xu, Moti Yung, and Haibo Zhang, editors, *Network and System Security*, volume 9408 of *LNCS*, pages 368–375. Springer, 2015.