Verifiable Secret Sharing Mechanisms - A Survey

Voggenreiter Markus
Betreuer: Dr. Matthias Wachs
Seminar Innovative Internettechnologien und Mobilkommunikation WS16/17
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: markus.voggenreiter@tum.de

ABSTRACT
In this paper, we show the state of the art technology in key escrow mechanisms. Secure communication over the internet as well as inside an intranet is based on the encryption of sent data. Hence the keys used for encryption are not only precious for the user but also for economical institutions. As soon as these keys are inaccessible the encrypted data can not be recovered, which might result in a loss of essential information. Therefore a way to regain these keys is fundamental for any institution. In this context Key Escrow provides a simple way to provide this desirable infrastructural feature. This paper will introduce several mechanisms to escrow keys and try to give a suitable solution for modern networks. Consequently key escrow in general will be explained and the currently popular mechanisms will be shown divided by the approach used in the infrastructure. These approaches will be explained on protocol layer and checked for the practicability. Finally it should give a conclusion over all protocols.

Keywords
Key Escrow, PKI, IBE, RIKE, CARIBE

1. INTRODUCTION
A fundamental part of modern communication is the security of communication. Therefore almost every entity communicating over the internet or any insecure channel uses cryptography to encrypt their information. Except for the positive outcome of pseudosecure communication, this trend also generates new issues. Assuming that any kind of key was used for protecting the data, all participating entities have the same preconditions for encryption and decryption of data. While the communication is protected, the generated data can only be read with the key intended to do so. This creates a long term problem with accessing all kind of data after the user key is lost. Especially when the data is critical, regaining it is essential.
In an institutional environment, adding, changing and deleting keys is a daily procedure. As soon as a key gets deleted, even though still necessary or an employee leaves, without transferring the decryption keys, the data encrypted with these keys is irrecoverably overwritten. Especially universities or high-tech industry deal with critical data and make use of cryptography for messaging and data transfer. The consequences of not having the ability to decrypt the data again, can result in a loss of money, time but also in legal actions. This problem can be solved by setting up key recovery or key escrow during the generation of such a key. This means the keys can be recovered after they are lost and therefore the encrypted data can be recovered. There are several approaches, which escrow the keys, based on, whether the keys are symmetrical or asymmetrical, which kind of key management is used and which special properties are expected.
Key Escrow in general has quite a bad reputation, since the expression also refers to a means to an end to give governments access to encrypted communication of their inhabitants. Already in 1990 the US government proposed the 'Clipper Chip', which generated a user key from a built in master key and sent the halved user key to two entities [1] . This enabled the access to encrypted data for the relevant US authorities. In modern escrow the underlying principle remained the same, but the Use Case changed. The use of key recovery/escrow in companies is not based on surveillance but on the ability of decrypting critical information encrypted with no longer existing keys.

In this paper several Key Escrow mechanisms will be discussed with focus on asymmetrical encryption in decentralized infrastructures. These will be rated based on several predefined requirements.

2. DEFINITIONS AND REQUIREMENTS
First of all several basic crypto mechanisms and encryption standards must be defined. Modern key management systems base on two major structures, the symmetrical and the asymmetrical encryption. These will be described as well as the criteria, used to benchmark the later presented mechanisms.

2.1 Symmetrical Encryption
Symmetrical encryption is a method to encrypt data sent between two entities with a single key. For this purpose the entities agree on a key first, which later is used to encrypt and decrypt the messages. An example for this would be the blockcipher AES [2], where the data is encrypted with in several rounds with the same key. Since this kind of encryption uses one key for every communication partner, it is not common for a scalable network. Due to this the work focuses on the usage of asymmetrical encryption.

2.2 Asymmetrical Encryption
The other encryption method, the asymmetrical encryption, is in terms of key management contrary to our first method. This technique uses two keys per user, where one key is used

doi: 10.2313/NET-2017-05-1_08

to encrypt data and the second key to decrypt data. In contrast to the symmetrical encryption these two keys are sufficient to communicate with all entities inside the system, since the encryption key of another person is used to encrypt data meant for him. The asymmetrical encryption is the common method for key management in larger systems. For the understanding of this paper the knowledge of basic asymmetrical encryption standards is necessary. Therefore the state-of-the-art techniques RSA and ElGamal will be presented.

2.2.1 RSA

RSA is based on the factoring problem, which results from the generation of the keys[3]. Every entity in the system owns two keys a private key (SK_{entity}) and a public key (PK_{entity}), which are used to encrypt and decrypt any message sent to them. In order to do so, the public key of the user, who should be able to decrypt the message/data must be used. These keys are derived from the following algorithm.
1. $n = p * q$ with $p \neq q$ and p, q are prime
2. $\varphi(n) = (q-1) * (p-1)$
3. e : coprime to $\varphi(n)$ with $e < \varphi(n)$
4. $e * d + k * \varphi(n) = 1$ and derive d from it
5. $PK_{user} = (n, e)$ and $SK_{user} = (n, d)$

The encryption is done with the public key:
1. $ENC(m) = m^e \ mod \ n = c$

The decryption with the private key:
1. $DEC(c) = c^d \ mod \ n = m$

2.2.2 ElGamal

ElGamal is based on the discrete logarithm problem, since it is based on the idea of the Diffie-Hellman protocol. It also relies on the basic idea of private and public keys, but these are generated differently than in RSA.
1. G of order q with generator g
2. $f \in \{1, ..., q-1\}$ with $ggT(f, q) = 1$
3. $h = g^f$
4. $PK_{user} = (G, g, q, h)$ and $SK_{user} = (f)$

The encryption is done with the public key:
1. $m' = m \in G$
2. $y \in \{1, ..., q-1\}$ with $ggT(h, y) = 1$
3. $c_1 = g^y$
4. $c_2 = m' * h^y$
5. $c = (c_1, c_2)$

The decryption with the private key:
1. $c_3 = c_1^f$
2. $m' = c_2 * c_3^{-1}$ with c_3^{-1} as inverse of c_3 in G
3. $m \Leftarrow m'$ with $\Leftarrow$ as reconversion

The ElGamal algorithm is less common in modern system than RSA. Therefore this paper will focus on infrastructures with RSA keys to escrow.

2.3 Requirements

The usage of Key Escrow comes along with several problems and unwanted features. To provide a fair comparison of the different mechanisms the must-have requirements and the good-to-have requirements are presented and explained.

Must-have:
1. The keys may not be sent in clear text. Otherwise it could be accessed by an attacker listening to the network.
2. It must be impossible for the user to use another key for decryption than the one escrowed. The user might send no key at all or doesn't use the key, which makes the escrowed key invalid.
3. It must be possible to generate new keys effortless. This means a new key should not force the whole system to reload.

Good-to-have
1. The key should be stored on more than one entity to reduce the risk of an entity cheating and decrypting the data. This also reduces the risk of a system loss, when one entity is infected.
2. A proof should be generated for the escrowed key, so that every user can verify the recoverability of the key.
3. The mechanism should be capable of being integrated into an existing system.
4. There should solely be the entity in need to escrow and the entity storing the keys. No third party should be required.
5. The entity storing the keys should not able to impersonate the user storing the keys. Since the user stores his private key, used to sign messages the entity is able to sign any message as the user with the private key.
6. Keys should be exchangeable effortless. Whenever a key is compromised the key pair of the user must be replaced. This should be doable as effortless as possible.

3. KEY ESCROW IN CPKIS

The main property of centralized public key infrastructures (CPKIs) is the centrally generated key pair. Whenever a user requires a key the central entity has to generate a key pair and send it to the user. This has to happen over a secure channel, to ensure the integrity and confidentiality of the keys. The key escrow in a CPKI happens directly at the central entity generating the key, depending on how the key was generated.

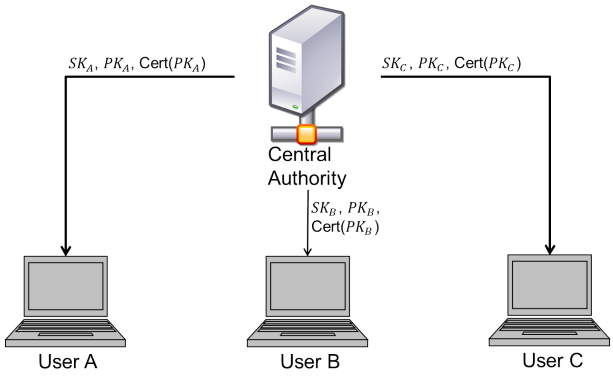

Figure 1: Key distribution in CPKIs

3.1 Private Key Storage

When generating an asymmetric key pair, the relevant part for decrypting data is the private key. Therefore the storage of each user's private key can be a solution. In this scenario the necessary entities are the key database and the key generator. The keys are generated and sent to the database as well as the user. These must be done over a secure channel to provide confidentiality and keep the key secret. In this scenario the key escrow fully relies on the database and the

doi: 10.2313/NET-2017-05-1_08

connection to it. Thus the database is a valuable target, for an attack.

3.2 Master Key Storage

When a master key is in use, the keys are getting computed based mainly on an identifier of the user and the master key. Therefore the key pair can be recovered by re-computing the user keys. This provides a way of key recovery, where only the master key must be escrowed.

3.3 Problem with Key Escrow

The main problem of key escrow in a CPKI is the access of the keys. Since the keys are stored at a single entity, everybody having administrative rights on it has normally access to the keys. Further accessing the keys does not require the cooperation of several entities, which would intercept on a cheating entity, wanting to access the key on its own, but since only one entity can access the key storage, a single entity is sufficient to gain the plaintext of all messages encrypted with the associated keys. Since the central authority generates all keys, it can impersonate any user in the system and read any encrypted message, without the user noticing.

4. KEY ESCROW IN DECENTRALIZED PKIS

Decentralized public key infrastructure(DPKIs) allow every user to generate his own key pair. This makes it, compared to a centralized PKI more flexible and usable, since the users are independent of a key generator and only need to send the public key to the centralized authorities in order to enable the certification, while the private key of the user stays with him. Whereas key escrow in a decentralized PKI is difficult, since the private key must be transmitted to one or more entities in order to escrow it. Therefore in DPKIs no escrow mechanism is part of the basic protocol.

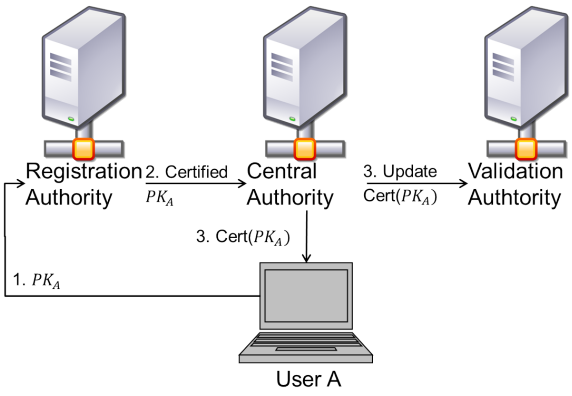

Figure 2: Key distribution/Certification in PKIs

4.1 Fair Encryption of Keys

A practical approach to a publicly verifiable proof that an entity involved in our cryptosystem is able to recover the secret key of our user, was proposed by Poupard and Stern [4]. In their scenario every user owns a pair of keys, from which the secret key is sent to an entity, that is not involved in process of verification or encryption [4, p. 172]. In this context the term "fair encryption" was used for this cryptosystem. Since their protocol operates on asymmetric encryption, especially with modern RSA, it can be used in a decentralized PKI.

4.1.1 Preconditions

To use the proposed "fair encryption" on modern RSA keys several cryptographical assumptions are made. First of all the entity used to store the secret key has a public key pair (N,G) in the Paillier cryptosystem [5]. Second user 1 has a RSA key pair (SK,PK) and wants to escrow SK. Since the user created SK and PK as RSA keys from $n = p * q$ with p and q as prime numbers (usually they must be different, but this protocol does not check the dissimilarity of them). A collision-resistant hash function like H()=SHA-256 must be available on all clients for the proof.

4.1.2 Key Escrow

The conditions for a fair encrypted secret key SK are furthermore a ciphertext Γ and a proof of fairness[4, p. 178]. First the user computes
$x = n - \varphi(n) = p + q - 1$,
where $\varphi()$ is Euler's totient function. To make the encryption probabilistic a random number should be added into the ciphertext. Therefore a random number $u \in \mathbb{Z}_N{}^*$ is selected. Now the Γ is calculated the as follows:
$\Gamma = G^x * u^N mod\ N^2$
Since the public RSA key of our user consists of the RSA modulus n and a number e coprime to $\varphi(n)$, the entity already knows our modulus n. Now the user sends Γ to the entity. The combination of n, Γ and the Parllier secret key of our entity can factor n in q and p and from these factors the private key SK from our user can be computed via
$e * d + k * \varphi(n) = 1$, solve for d[4].

4.1.3 Proof of validity

A main characteristic of this protocol is the non-interactive key storage entity. This means any validation is done by the user itself. The typical proof consists of user 1, who wants to proof the key and user 2, who wants to see, whether the key can be recovered. Thus both users first have to agree on randomly chosen integers
$z_i \in \mathbb{Z}_n^*$ for i=1,...,K.
Then user 1 has to compute the proof:
Choose:
$(r_i)_{i=1,...,l} \in_R [0, A[^l$ and $(v_i)_{i=1,...,l} \in_R \mathbb{Z}_N{}^{*l}$
Generate:
$t = ((G^{r_i} v_i^N mod N^2)_{i=1,...,l}, (z_j^{r_i} mod n)_{i=1,...,l;j=1,...,l}$ and
$(e_1, ..., e_l) = H(t, N, G, (z_j)_{j=1,...,K}, n)$ and $\{\ y_i = r_i + e_i * x$ and $y_i' = u^{e_i} * v_i \mod N\ \}$ for $i = 1, .., l$
Now user 1 sends the proof $((y_i, y_i', e_i)_{i=1,...,l})$. This proof is verified by user 2:
Check $0 \leq y_i < A\ for\ i = 1, ..., l$
Generate:
$t' = ((G^{y_i} * y_i'^N / \Gamma^{e_i}\ mod\ N^2)_{i=1,...,l},$
$(z_j^{y_i - e_i n}\ mod\ n)_{i=1,...,l;j=1,...,K})$
Check $(e_1, ..., e_l) = H(t', N, G, (z_j)_{i=1,...,K}, n)$
If the equality of the last line is given, the key is recoverable by the entity with N and G as public key[4, p. 183].
In this proof several variables were used. Poupard and Stern made a proposal for them in the year 2000. Since then the recommended key sizes increased to a minimum of 2048 [6], these sizes should be adapted. Since we use SHA-256 as

doi: 10.2313/NET-2017-05-1_08

our hash function the values of e are between 0 and 2^{256} . The probability of success in breaking the protocol is $1/B^l$, which would be in our case $1/2^{256l}$. That makes the length l=2 proposed in 2000 still usable. The variable A is a number smaller than n and larger than $x * 2^{256} * 2$. Since the key length of RSA should be 2048, $A < 2^{2048}$. Thus a number greater 2^{1300} can be sufficient. In case of K the used integer has no real effect on the security. In 2000 the value 3 was proposed and since it only changes the amount of random numbers used in the protocol any number higher than 5 should be sufficient[7, compare with].

4.2 Auto-recoverable Cryptosystems

Another way of escrowing keys with a verifiable proof was presented in the year 2000 by Young and Yung. They dealt with the topic of auto-certifiable and auto-recoverable cryptosystems [8] in general and published an implementation of it based on ElGamal [9] in 1999. This was expanded on RSA in 2000. This cryptosystem escrows the private RSA key of a user and gives proof on the recoverability. This scheme works on one escrow agent as well as on a group of escrow agents, since it basically just encrypts the private key. The protocol consists of four steps. The first one is the generation of the RSA key pair. The second one is the generation of the certificate, which provides the data for the recovery to the escrow agents and a proof for anybody else. The third one is the verification of the certificate and the fourth one the recovery of the private key by the recovery agents. In this example the non-interactive version of this protocol is presented.

4.2.1 Preconditions

For the initialisation, the generation, the verification and the recovery of the key several functions need to be defined. These variables might be initialized during the setup of the system and give an overview.
ENC(r,s,E) encrypts the plaintext r using the random value s with the key E.
H(x) is a ideal hash function, hashing x with range Z_n^*.
H"(x) is a random oracle hash function, hashing x.
"E_i" is the public key of the escrow agent i.
"e" is one part of the public key of the user, that wants his key escrowed.
"n" is the RSA module of the public key.
"d" is the private key of the user, that needs to be escrowed.
"$a_i \in_R Z_{\varphi(n)}$" is a random number, chosen by the user.
"$s_{i,1}$ *and* $s_{i,2}$" are the two random numbers for the encryption, chosen by the user.
"t_i *and* t_i'" are variables used during the certification. They either are generated randomly or given by the verifier.
"P" is the computed proof by the user.
$\lambda(n)$ is the Carmichael function.
"m" is the amount of escrow entities.
"k(m)=$\omega(log\ m)$" is the length of the transcript.
"δ" is an additional variable to increase the size of the transcript independent of m [10, p. 329-334].
For the key recovery it is essential, that n consists of two different prime numbers. As protocol to validate this Young and Yung propose [12].

4.2.2 Generation

In this step the RSA keys are created. This is the standard process of RSA key generation, where n is the product of two distinct prime-numbers p and q, e is a number coprime to $\varphi(n)$ and larger 0 and d is the modular multiplicative inverse of $e(mod\varphi(n))$.[10, p. 332] Afterwards the variables e, n and d are initialized.

4.2.3 Key Certification

To construct the proof of escrow, we need the RSA key pair of our user. The output is a proof P, with which the escrow can be verified.[10, p. 330]
1. Initialization
First the initialization takes part, then some randomly based values are generated:
$P = (n), t_0 = H(n)$
$for\ i = 1\ to\ \delta k(m)$:
$t'_{i-1} = H(t_{i-1})\ ;\ t_i = t'^{e}_{i-1}\ mod\ n$
Now the encryption of the data, later relevant for key recovery is done and added to the end of P. In this example the key used for encryption is just E. This could also be a set of keys from several escrow entities $E_1, ..., E_m$[10, p. 331]:
$for\ i = 1\ to\ \delta k(m)$:
$a_i \in_R Z_{\varphi(n)}$;
$s_{i,1}$ *and* $s_{i,2}$ randomly chosen;
$v_i = t_i^{a_i}\ mod\ n$
$C_{i,1} = ENC(a_i, s_{i,1}, E)$;
$C_{i,2} = ENC(d - a_i\ mod\varphi(n), s_{i,2}, E)$;
$add(v_i, C_{i,1}, Ci, 2)$ to P [10, p. 332].
2. Partially Proof
At last the actual "proof" is generated, since the user needs the following to verify the correct encryption of the data, which is made probabilistic by a random oracle.
val = H"(P);
set $b_1, ..., b_{\delta k(m)}$ as the least significant bits of val ($b_i \in \{0, 1\}$)
$for\ i = 1\ to\ \delta k(m)$:
$a_{i,1} = a_i; a_{i,2} = d - a_i\ mod\ \varphi(n)$; $z_i = (a_{i,j}, s_{i,j})$ with $j = 1 + b_i$
add z_i to the end of P
3. Entire Proof
Now the entire proof is generated and has the form:
P = $(n, (v_1, C_{1,1}, C_{1,2}), ..., (v_{\delta k(m)}, C_{\delta k(m),1}, C_{\delta k(m),2})$, $z_1, ..., z_{\delta k(m)})$ [10, p. 333]

4.2.4 Verification

Since the verifier knows
P = $(n, (v_1, C_{1,1}, C_{1,2}), ..., (v_{\delta k(m)}, C_{\delta k(m),1}, C_{\delta k(m),2})$, $z_1, ..., z_{\delta k(m)})$ where $z_i = (a_{i,j}, s_{i,j})$ with $j = 1 + b_i$
he can extract n from P. This implies he can generate $t_1, ..., t_{\delta k(m)}$ from the known public key e and the hash function H(), similar to the key certification. Further he can compute $b_1, ..., b_{\delta k(m)}$,
as he can hash the first part of P:
$P_2 = (n, (v_1, C_{1,1}, C_{1,2}), ..., (v_{\delta k(m)}, C_{\delta k(m),1}, C_{\delta k(m),2}))$ with the function H"(). Now several values are checked:
First of all some general values need to be verified. These are the basic variables used for the generation of the proof.
$t_i \in Z_n^*$ and $a_{i,1+b_i} < n$ for $1 \leq i \leq \delta k(m)$.
Now the actual verification takes place.
$C_{i,1+b_i} = ENC(a_{i,1+b_i}, s_{i,1+b_i}, E)$,
where the values from P_2 are compared with z_i.
$t_i^{a_{i,1+b_i}} = (t'_{i-1}/v_i)^{b_i} v_i^{1-b_i}\ mod\ n$,

doi: 10.2313/NET-2017-05-1_08

where $t_i^{a_{i,1+b_i}}$ can be computed from our first step of verification and the first part of z_i. The second part of the equation is given either just by P_2 or by P_2 and t_i'. As long as these equations and the basic variables are correct the key is correctly escrowed. [10, p. 333].

4.2.5 Recovery

First of all the escrow agents need to decrypt $C_{i,1}$ and $C_{i,2}$, to extract the corresponding plaintext either a_i or $d-a\ mod\varphi(n)$. Now d_i' is computed as the plaintext from $C_{i,1}$ plus the plaintext of $C_{i,2}$. Further K_i is generated via $K_i = ed_i' - 1$. In this way K_i is created for each i from 1 to $\delta k(m)$ [10, p. 333]
Now it is possible to factor n given K_i with a Las Vegas algorithm [11, p. 10]. This algorithm is run on K_i for all i.

5. IDENTITY BASED ENCRYPTION

The basic idea behind IBE, abbreviation for Identity Based Encryption, is to encrypt messages with an identifier instead of a key. This scheme differs in several points from a PKI, but the main advantage of an IBE scheme is the built in escrow mechanism. Therefore the usage of IBE instead of PKI to enable key escrow is a noteworthy option.

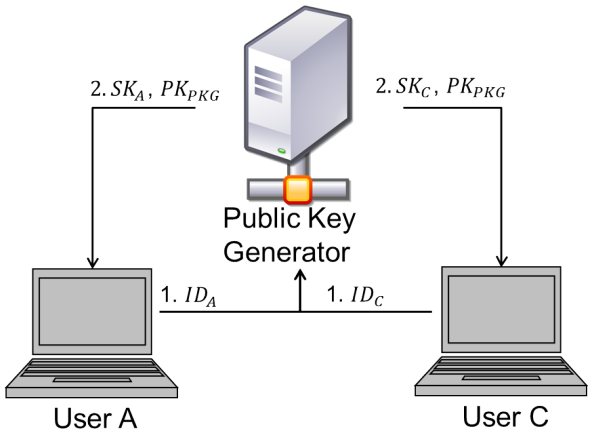

Figure 3: Key distribution in IBEs

5.1 IBE in General

The basic IBE scheme consists of an amount of users n and a private key generator PKG [13]. Each user has an identifier the so called ID. Based on this ID the user can request a corresponding private key from the PKG. The PKG owns a private and a public key, whereas the public key is rather a set of system parameters. For the usage of the encryption scheme (IBE.E) the following four basic algorithms are the framework as described by Boneh and Franklin in 2003:
1. Setup, which the PKG runs once to derive the public and the private key (MK) from a security parameter k, where the public key is generally called "params" meaning system parameter.
2. Extract, which is run by the PKG each time a new user requests a private key and giving the PKG his ID. It takes MK, params and the user's ID to generate a corresponding private key (d), that is given to the user.
3. Encrypt, which is run each time a user wants to send a message encrypted. This algorithm takes the ID of the receiver, the message to send and params to encrypt the message.
4. Decrypt, which the receiver of an encrypted message must run, to decrypt it. This takes the encrypted message, the ciphertext, params and d of the receiver and outputs the plaintext message.[14]
As one may notice the critical spots are the connection between the user and the PKG and the risk of a compromised PKG.

5.2 Key Escrow in IBEs

One of the main features of identity based encryption is the automated key escrow. Since the user gets his private key from the PKG, the PKG has two different strategies how to escrow the private keys of the user.

5.2.1 Key Escrow with Master Key

Due to the fact, that the PKGs private key (MK) is used to generate the users private key, while as only other variables the static values params and ID are used, the private key can be recovered by running the algorithm again. This strategy is the most comfortable, since the amount of data to store and the effort to restore a key is minimum. On the other hand the MK is the single point of failure in the escrow system. When this key is lost, the whole system beneath this PKG can not be escrowed anymore, aside from the fact that if it was stolen, the attacker has full control over the system.

5.2.2 Key Escrow by Storing Keys

To assign the key escrow to a third entity, an option is to store the computed private keys of the user in a database, which is not necessarily on the same system as the PKG. This increases the required storage capacity, keeps the effort at a minimum and distributes the critical data, seen from the side of key escrow, on two systems.

5.2.3 Problems with Key Escrow

The main problem concerning key escrow in IDE schemes is, the access of escrowed keys. Since in both shown ways of escrowing keys, these keys can be accessed by a entity and do not require a cooperation of several entities(compared to PVSS mechanisms). So a special kind of four-eyes principle must be implemented if this kind of security is required.

5.3 Cascade-realized Identity Based Encryption

The basic protocol of IBE consists of one single encryption scheme, used on one PKG. As seen earlier this creates several problems with the security of the key escrow. This issue could be solved by splitting the PKG authority into a group of entities, which was proposed as a cascade realized identity based encryption scheme(CARIBE) by Hale, Carr and Gligoroski [15].
In this scheme n different PKGs cooperate to generate keys, encrypt and decrypt the messages based on possibly n different encryption schemes. This is realized by encrypting the messages under a cascade realisation, where the ciphertext of the first encryption is the plaintext of the second encryption and so on.

doi: 10.2313/NET-2017-05-1_08

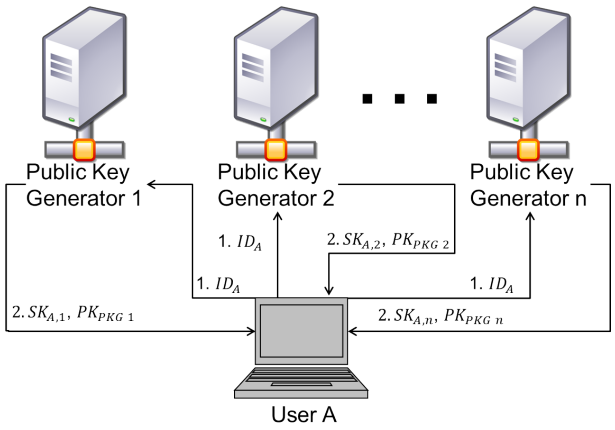

Figure 4: Key distribution in CARIBE

5.3.1 Algorithms

The algorithms are based on the same framework as described in [5.1], but differ in the amount of variables used in every step. Further there can be n different encryption schemes, used during the setup[15, p. 8-9]. The used function calls refer to the specific function used in scheme i, e.g. $Setup_i$ is the function $Setup$ of the encryption scheme i:

1. Setup, where n security parameter (k) are taken and n params and master keys MK_i are generated:
$for\ i \in \{1, ..., n\}$:
$(params_i, MK_i) = Setup_i(k_i)$ end
return $(params_1, ..., params_n\}, \{MK_1, ...MK_n\})$

2. Extract, which takes n params and MKs and the ID of one user and computes the corresponding n private keys d_i:
$for\ i \in \{1, ..., n\}$:
$d_i\ = Extract_i(params_i, MK_i, ID)$ end
return $d_1, ..., d_n$

3. Encrypt, where message m is encrypted using the ID of the user and n params to generate the ciphertext:
$c_2 = Encrypt_1(params_1, ID, m)$
$for\ i \in \{2, ..., n\}$:
$c_{i+1} = Encrypt_i(params_i, ID, c_i)$
end
$c = c_{n+1}$; return c

4. Decrypt, where the ciphertext, n private keys of the receiving user (d_i) and n params are used to produce the plaintext:
$c_n = c$; i=n
$for\ i > 0$:
$c_{i-1}\ = Decrypt_i(params_i, c_i, d_i)$; i=i-1
end
$m = c_0$; return m

As one can see the messages are cascaded en-/decrypted with n keys, n params under n schemes(not necessarily different).

5.3.2 Problems with Key Escrow

The main problem mentioned in [5.2.3] was the access of the keys. This problem has been solved by giving each PKG only access to one layer of encryption. If using the MK of each PKG, n different entities must be requested to decrypt the message. Otherwise if the keys are not stored in a single database, also n databases must be requested for the access. Solely if all user keys are stored in one database the problem keeps remaining. The only missing property would be the ability to regain access to the encrypted data, even though $PKG_x \subset \{PKG_1, ..., PKG_n\}$ is compromised or the keys are lost.

5.3.3 Problems with CARIBE

There are several problems going along with a cascade realisation of IBE. First of all the length of the ciphertext increases, since several encryptions are run successively. This problem was declared as not imposing, as modern IBE schemes are less expanding and the transmission rates are fast enough to deal with this size of data[15, p. 12]. Another problem is the new amount of keys. These must be stored securely at the client, but first of all need to be transmitted to the client. The usage of CARIBE increases the amount of necessary secure channels to transmit data in advance. Since IBE at all does not deal with the problem of key distribution, this is no CARIBE specific problem. Further the performance of en- and decryption is problematic, since these must be run n times. At last cascade cryptography is seen as CCA insecure, as any user can decrypt and re-encrypt the outermost encryption, when in possession of the key. But the possession of this key is only an assumption and not realistic for this case [15, p. 9].

5.4 Risks of IBE

IBE on its own has few considerable security risks. The connection between user and PKG must be highly secure, since the private key of the user is sent over this channel. This general problem of PKG can only be solved based on the use case. Also a compromised key is a problem, since the identifier is the public key and therefore the identifier of this user must be changed in order to grant security, but these identifiers are normally static values(like email addresses). Since the PKG generates all keys, it can on the one hand impersonate every user in his system and on the other hand can decrypt all messages. Because of these enormous privileges, the PKG is a worthwhile attack target. As soon as the PKG is compromised the whole system is compromised and can no longer be used.

5.5 RIKE

A way to integrate IBE into PKI was proposed in 2012[16]. The key management infrastructure was called RIKE and integrates a identity based encryption into a already existing PKI to enable key escrow in the PKI. This is basically done by hashing the user certificate and the value computed in this way is the new public key of our user in the IBE scheme.

5.5.1 Algorithm

The preconditions for implementing RIKE is an existing PKI with the user key pair (PK_u, SK_u) for every user and an CA with identity ID_{ca} and keys, which signs the users public key and returns a certificate $Cert(U) = SIGN_{SK_{ca}}(ID_{ca}, PK_u)$ signed by the private key of the CA. The integration of the IBE requires several values, which are equal to the data used for an IBE. This comprises the PKG, with the system parameters params, the private key of the generator MK and the standard functions Encrypt, Decrypt and Extract. These function are the same as in a basic IBE scheme[16, p.

52].
1. Initialization, where params and MK is generated, similar to [5.1/1.] and signs the new certificate $CERT(CA, params) = SIGN_{SK_{ca}}(ID_{ca}, PK_{ca})$ and adds params as extension[16, p. 62/63]. This certificate is sent to every user, so that every user has params and the certificate of the CA.
2. Key Generation, where the PKG produces the second private key of the user from the hash of the users certificate $SK_{u2} = Extract(MK, params, H(Cert(U)))$ comparable to [5.1/2.]. Now the second, escrowed secret key gets sent to the user. The user now has (SK_u, PK_u), which is the not escrowed pair and the new $(SK_{u2}, (PK_{u2} = H(Cert(U))))$ as the escrowed key pair.
3. Usage, can be done from now on, since the IBE scheme is now implemented. The encryption and decryption works just as in a normal IBE scheme with the escrowed keys, except for the fact that the certificate of the user is checked first. This grants a fast way of revoking compromised keys in the IBE scheme. Also signing can be realized by using the not escrowed private key (SK_u). That solves the problem of the almighty PKG, because it can no longer impersonate any user of the system.[16, p. 52/53]

5.5.2 RIKE based on different structures

The larger the cryptographic environment is, the more entities are needed to manage the users. This tends to result in a increasing number of CAs. To deal with this, RIKE is also compatible to a hierarchical CA structure in [16, p. 54-56]. Usually there are several different CAs cross-signing each other in order to ease the validation of certificates for users. This can also be implemented by RIKE, which is described in [16, p. 56-59].

5.5.3 Key Escrow in RIKE

RIKE builds an IBE inside a PKI, so the key escrow properties apply to the system. But this only applies to the keys generated by the PKG, not to the existing asymmetrical keys. Therefore the use of the first key pair has to be forbidden for encryption to enable full escrow.

6. OTHER KEY ESCROW OPPORTUNITIES

There are further ways of escrowing keys in different cryptosystems. Since every of these schemes are comparable complex to the others given in this paper, they will just be explained briefly.

6.1 SE-PKI

Self escrowing public key infrastructures deal with key escrow by intervening into the key generation at the user. The user generates his key pair decentralized, but with the public part of trapdoor information provided by a central entity. This connects the public and the private key under a master scheme. Then he computes a proof, which everybody can verify. In case of a lost key, the users private key can be recovered by the private part of the trapdoor information by a central authority. The key recoverability can be distributed between several entities.[17]

6.2 PVSS

A rather universal approach are the publicly verifiable secret sharing mechanisms. These mechanisms allow to share a secret among several participants and generates a proof for the recoverability of this secret. [19] This proof can be verified by any user of the system. As mentioned in [18] the usage of these properties enable key escrow in any infrastructure. Instead of a secret, the private key of the user is divided into shares and sent to the escrow entities. Any user of the system can verify the recoverability of the key. As soon as the key must be recovered the entities having the shares must collaborate to recover the key. This is a quite universal solution to the problem of Key Escrow.

7. EVALUATION

This paper presented a solution for different key infrastructures. At last these mechanisms will be compared and rated based on our requirements. The first must-have can not clearly be fulfilled by any mechanism. The transmission of keys must always be done in a secure way, whether using encryption or manual techniques. Solely the Fair Encryption and the auto-recoverable cryptosystems directly refer to this problem, as they send the encrypted private key to the central entity. Therefore this requirement will be seen as a challenge, which must be solved by other means.
1. CPKI
The first two mechanisms used in a centralized key infrastructure fulfil all must-haves, but not all other requirements. They fail in 1, 3 and 5. Since the keys are solely generated centrally 2 is not required. Further the keys can only be exchanged by requesting such a change, but there are no major changes required in the system. All in all this technique fulfils most of our requirements, but the required infrastructure makes it highly impractical and hence limited useful.
2. Fair Encryption
The Fair encryption fulfils all must-haves, since a valid proof requires the usage of the public key. This protocol fails in 1 and 5. Especially the non-interactive proof executed solely by the users of the system makes this protocol convenient. The only drawback is the usage of the Pallier cryptosystem, which is not supported by every system. Therefore this can be integrated into any system, but only with mentionable effort, which makes this method a possible but not the best solution.
3. Auto-Recoverable Cryptosystem
The Auto-Recoverable Cryptosystem also fulfils all our must-haves. Since it is possible to increase the number receiving entities, all of the optional requirements can be fulfilled. Since it also does not require any specific encryption it can be integrated into any system. Since the protocol itself is more than 15 years old, the protocol should be investigated for possible flaws before using it.
4. IBE
The next protocol is the usage of IBE properties. This Escrow mechanism is similar to the first two mechanism, since it uses the centralized generation to escrow the key. So it actually does fulfil all must-haves, but fails in 1, 3, 5 and 6. Changing keys is complex, since the system is based on identifiers. Further it is a completely different infrastructure than the common PKI, which makes it incompatible.
5. CARIBE
The IBE based protocol CARIBE solves several problems of this IBE scheme. It is conform to all our must-haves and

also achieves more optional properties than IBE. It only fails with 3, since the distribution of the keys on different entities solely leaves the problem of compatibility.

6. RIKE

This problem can be solved by the last protocol RIKE, which complies all must-haves, but fails with 1. It gains the compatibility by using an existing PKI, which leaves the problem of a single entity controlling the keys.

8. CONCLUSION

Key escrow in a nutshell is a very complex topic. Not only that most studies try to get rid of key escrow instead of improving it, but modern key escrow often relies on a lot assumptions. On the other hand most key escrow mechanisms can be integrated in every key scheme and is therefore highly usable. Any presented algorithm can be integrated in a public key infrastructure, which is the common cryptosystem. A suitable solution for a PKI system, would be the integration of RIKE in combination with CARIBE. Most mentioned problems would not appear in this line-up and the solely drawback is the management of the different entities and keys.

9. REFERENCES

[1] Han-Wei Liu. 2016. Inside the Black Box: Political Economy Behind the TTP's Encryption Clause, Journal of World Trade, pages 13/14.

[2] Joan Daemen and Vincent Rijmen. 2002. The Design of Rijndael. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[3] R. L. Rivest, A. Shamir, and L. Adleman. 1978. A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM 21, 2 (February 1978), 120-126. DOI=http://dx.doi.org.eaccess.ub.tum.de/10.1145/359340.359342

[4] Guillaume Poupard and Jacques Stern. 2000. Fair encryption of RSA keys. In Proceedings of the 19th international conference on Theory and application of cryptographic techniques (EUROCRYPT'00), Bart Preneel (Ed.). Springer-Verlag, Berlin, Heidelberg, 172-189.

[5] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In Proceedings of the 17th international conference on Theory and application of cryptographic techniques (EUROCRYPT'99), Jacques Stern (Ed.). Springer-Verlag, Berlin, Heidelberg, 223-238.

[6] Cryptographic Key Length Recommendation, `https://www.keylength.com/en/8/`, 18.12.2016.

[7] Guillaume Poupard and Jacques Stern. 2000. Short Proofs of Knowledge for Factoring. In Proceedings of the Third International Workshop on Practice and Theory in Public Key Cryptography: Public Key Cryptography (PKC '00), Hideki Imai and Yuliang Zheng (Eds.). Springer-Verlag, London, UK, UK, 147-166.

[8] Adam Young and Moti Yung. 1999. Auto-recoverable Auto-certifiable Cryptosystems (A Survey). In Proceedings of the International Exhibition and Congress on Secure Networking - CQRE (Secure) '99, Rainer Baumgart (Ed.). Springer-Verlag, London, UK, UK, 204-218.

[9] Young, Adam L. and Moti Yung. Auto-Recoverable Cryptosystems with Faster Initialization and the Escrow Hierarchy. Public Key Cryptography (1999).

[10] Adam Young and Moti Yung. 2000. RSA-Based Auto-recoverable Cryptosystems. In Proceedings of the Third International Workshop on Practice and Theory in Public Key Cryptography: Public Key Cryptography (PKC '00), Hideki Imai and Yuliang Zheng (Eds.). Springer-Verlag, London, UK, UK, 326-341.

[11] Hinek, M. Jason. 2009. Cryptanalysis of RSA and its variants. Boca Raton: CRC Press. url=https://books.google.de/books?id=LAxAdqv1z7kC,

[12] Z. Galil, S. Haber, M. Yung. Minimum-knowledge Interactive Proofs for Decision Problems. In SIAM J. of Computing, (4), pages 711?739, 1989.

[13] Clifford Cocks. 2001. An Identity Based Encryption Scheme Based on Quadratic Residues. In Proceedings of the 8th IMA International Conference on Cryptography and Coding, Bahram Honary (Ed.). Springer-Verlag, London, UK, UK, 360-363.

[14] Dan Boneh and Matthew Franklin. 2003. Identity-Based Encryption from the Weil Pairing. SIAM J. Comput. 32, 3 (March 2003), 586-615. DOI=http://dx.doi.org/10.1137/S0097539701398521

[15] Hale, Britta, Christopher Carr and Danilo Gligoroski. CARIBE: Adapting Traditional IBE for the Modern Key-Covetous Appetite. IACR Cryptology ePrint Archive 2015 (2015): 1035.

[16] Nan Zhang, Jingqiang Lin, Jiwu Jing, and Neng Gao. 2012. RIKE: using revocable identities to support key escrow in PKIs. In Proceedings of the 10th international conference on Applied Cryptography and Network Security (ACNS'12), Feng Bao, Pierangela Samarati, and Jianying Zhou (Eds.). Springer-Verlag, Berlin, Heidelberg, 48-65. DOI=http://dx.doi.org/10.1007/978-3-642-31284-7_4

[17] Paillier, P., Yung, M.: Self-Escrowed Public-Key Infrastructures. In: Song, J.S. (ed.) ICISC 1999. LNCS, vol. 1787, pp. 257?268. Springer, Heidelberg (2000)

[18] Adam Young and Moti Yung. 2001. A PVSS as Hard as Discrete Log and Shareholder Separability. In Public Key Cryptography: 4th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2001 Cheju Island, Korea, February 13–15, 2001 Proceedings. Springer Berlin Heidelberg.

[19] Markus Stadler. 1996. Publicly verifiable secret sharing. In Proceedings of the 15th annual international conference on Theory and application of cryptographic techniques (EUROCRYPT'96), Ueli Maurer (Ed.). Springer-Verlag, Berlin, Heidelberg, 190-199.