Distributed PKI in P2P Networks

Martin Schanzenbach
Betreuer: Matthias Wachs
Seminar Innovative Internettechnologien und Mobilkommunikation SS2010
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: schanzen@in.tum.de

ABSTRACT
Internet security today is based almost entirely on a public key infrastructure that allows authentication and encryption of data. However this PKI heavily relies on central institutions, namely the Certification Authorities (CAs), that issue certificates. In pure P2P networks such central instances are unwanted because they contradict the P2P paradigms. In this paper, we describe how a Certification Authority can be efficiently maintained and distributed across all peers in a P2P network. This makes it possible to put away with central CAs in P2P networks by establishing a *distributed* public key infrastructure (DPKI).

Keywords
P2P, PKI

1. INTRODUCTION
Today peer-to-peer (P2P) networks are becoming increasingly popular. They provide a lot of features that traditional client and server architectures simply don't, for example there is usually no single point of failure.

In common client and server architectures *authentication* is often realised through a *public key infrastructure* (PKI). PKI authentication today is usually done using X.509 certificates. Servers can identify themselves to the client by showing their certificate which needs to be signed by a *trusted third party*. Those *trusted third parties* are also called *Certification Authorities* (CA) because they have the ability to issue the certificates.

However the CA is a central instance in the network and thus opposes the idea of a pure P2P system. In this paper we try to solve this issue by distributing the functionality of the CA across all peers in a P2P network. In such a *distributed* public key infrastructure (DPKI) collaborating peers provide the functionality of the CA. Hence for example issuing a certificate depends on multiple peers working together. Also the single *trusted third party* mentioned above is represented by all the peers in the network.

This paper is structured as follows: At first we present some related work that has already been done in terms of distributed cryptography. In chapter 3 we discuss the basics of P2P networks. Chapter 4 deals with the cryptographic routines used to realise the distributed PKI. A small introduction of the PKI is given in Chapter 5. Chapter 6 then explains in detail the approach to set up a distributed PKI. Finally in chapter 7 a few possible attacks on this certification scheme are discussed.

2. RELATED WORK
Shamir proposes a method in [5] that allows to divide any data into n parts where the knowledge of k parts is enough to completely reconstruct the original data. Whereas knowing only $k-1$ parts will yield no information on the data whatsoever. The author called this a (k, n) threshold scheme that can be "very helpful in the management of cryptographic keys" [5]. This is particularly interesting in the scope of our work to distribute a public key infrastructure.

Another interesting work was done by Rivest et al. [4]. They present a *ring signature* scheme. This signature can be created by anyone of the same group, or "ring", with his own secret key. The signature can be verified with the group's public key, that is shared across all members. Hence it is not possible to identify the signer, which is an advantage in terms of privacy. However they do not propose a concrete implementation of their scheme.

Finally we are going to look at some related work regarding distributed certification. Zhou et al. [6] propose a distributed *Certification Authority* by the use of *threshold cryptography*. The authors present *Cornell On-line Certification Authority (COCA)*, a "fault-tolerant and secure on-line certification authority". This approach allows a ratio of servers that act as the CA to get compromised. As long as this ratio is within a certain threshold the certification process is still possible because enough servers can collaborate to create the certificate. This is possible due to the use of *threshold cryptography* which is explained in detail in section 4. However this approach only addresses client and server architectures and not P2P systems.

3. BASICS OF P2P NETWORKS
Peer-to-peer (P2P) networks are often used to design high available and low-cost systems. Putting away with the traditional client/server model, in which one system acts as a central instance (the server), the P2P network treats every node in the network equally. This means that participating peers can act as both – client and server – depending on the situation. There are two different kinds of peer-to-peer networks – structured and unstructured ones. Both will be discussed in the following.

3.1 Unstructured

When a peer requests data in an unstructured P2P network its request is flooded through the network. Such a technique creates a very high amount of signalling traffic [**?**]. An example for an unstructured P2P network is Gnutella. Today *structured* P2P networks are more common because they provide the basics for a more efficient network.

3.2 Structured

A structured P2P network allows any peer to efficiently look up data in the network. This is often realised through *distributed hash tables* (DHTs). DHTs serve a similar function like traditional hash tables. Peers can efficiently find data by using the DHT to look up a peer that owns the data. This works by querying the DHT for peers that can provide a specific data set. An example for such a network, that is using Kademlia DHTs, is the distributed tracking system of the popular BitTorrent protocol.

4. BASICS OF DISTRIBUTED ASYMMETRIC CRYPTOGRAPHY

PKI in general is based on asymmetric cryptography. The distributed approach for P2P networks we propose is no different. However the processes of *key generation*, *generation of signatures* and *encryption* differ slightly from the traditional non-distributed approach.

At first we are going to discuss the basics of asymmetric cryptography. Then we take a look at methods for key generation as well as signing and encrypting in a distributed fashion.

4.1 Asymmetric cryptography

Asymmetric cryptography is a method that can be used to sign and encrypt data. It is quite different to its symmetric counterpart. Instead of using a single secret key for encryption so called *key-pairs* (P, S) consisting of a **public** key P and a **private** key S are used. The public key P can be distributed through insecure channels whereas the private key S should always remain secret. To **encrypt** any kind of data **t** the public key can be used. **Decryption** is possible by applying the private key. RSA is a well known cryptosystem that can be used for asymmetric cryptography. It defines $P = (d, m)$ and $S = (e, m)$. In this case **e** is the secret part of the key pair, whereas **d** is the public part. The modulo $\mathbf{m} = \mathbf{p} * \mathbf{q}$ is required for the calculations when de- and encrypting and p and q are big prime numbers. This is why the security of the RSA cryptosystem is largely based on the mathematical problem of factoring large numbers. To encrypt any data **t** it is exponentiated with the secret **e**: $s = t^e[m]$. Decryption is done by exponentiating the encrypted data with the public part **d**: $t = s^d[m]$ An additional feature of RSA is called **signing**. To **sign** any data set the private key is used. Consequently **verifying** the resulting signature is possible with the public key. Anybody who knows the public is then able to verify this signature. Hence this feature unsuitable to ensure that the data cannot be read by a third party. But for example it is useful for identifying its origin.

4.2 Distributed RSA key generation

For asymmetric cryptography to be of any use in a distributed environment like a P2P network it is also necessary to think of a distributed way to generate the key pair (P, S). It is important that no peer in the network knows the secret key S entirely. So the peers have to jointly generate **shared** RSA keys. Fortunately efficient algorithms to do this already exist like the one proposed by [1].

4.3 Distributed signing and encryption using RSA

RSA has a crucial property: It is a homomorphism. Without this property it would not be possible to use it in a distributed environment. Using the distributed key generation algorithm introduced above all parties know only a part e_i of the secret e after the key pair is generated. However due to the homomorphic property of RSA it is possible to assemble partial signatures t^{e_i} into the full signature t^e. Equation 1 illustrates how the homomorphic property of RSA can be utilised. By applying basic rules of exponentiation it can be shown that instead of computing $t^e[m]$ it is also possible to compute $t^{e_i}[m]$ for every part e_i of the secret e and in the end multiplying the results. In other words it is possible to generate a signature without actually knowing the secret key S as long as all partial signatures are available.

$$t^e[m] = t^{\sum_{i=1}^{n} e_i}[m] = (\prod_{i=1}^{n} t^{e_i}[m])[m] \quad (1)$$

In fact it is not necessary to combine *all* partial signatures. It is sufficient to settle for a ratio or *threshold* **r** on key generation. To retrieve the full signature it is then enough to combine any **r**% partial signatures. This method is based on [5] and [2] and is accordingly called *threshold cryptography*. This feature can be used in a P2P network to sign data, or create certificates for that matter, without any party in the network actually knowing the secret key $S = (e, m)$.

5. PUBLIC KEY INFRASTRUCTURE

In a *Public Key Infrastructure* (PKI) certificates are issued by a *Certification Authority* (CA). A certificate can be used by the party that it was issued to, to authenticate itself by showing it to another party. Such a certificate is signed and issued by the CA. Hence if one party trusts the CA it can also trust the certificate that was issued to the other party. Today the PKI is often used in combination with HTTPS and SSL/TLS for secure interaction on websites with sensitive content. For example on-line-banking or web mail services take advantage of the PKI to authenticate themselves to the user to prevent fraud.

5.1 Certification Authorities

CAs issue digital certificates signed with their private key. If two parties, A and B, do not trust each other but one of them owns a certificate signed by a CA, this party can authenticate itself by showing this certificate. The other party can validate the certification by checking the signature with the CA's public key. Unless both parties have a certificate, though, this authentication can only go one-way.

Of course this only works in the first place if the CA is trusted by both parties. This is also called a *chain of trust* and the CA is often referred to as the *trusted third party* (TTP).

5.2 Disadvantages for P2P systems

In (*pure*) P2P networks there are no central instances. Every peer is treated equally. Consequently an institution like a CA contradicts this paradigm. To fully integrate the concept of a PKI into a P2P network this central CA needs to get distributed across all the peers that participate in the network. Illustration 1 opposes the central PKI and the distributed PKI (DPKI). In the following we propose an approach to build such a DPKI for a P2P network.

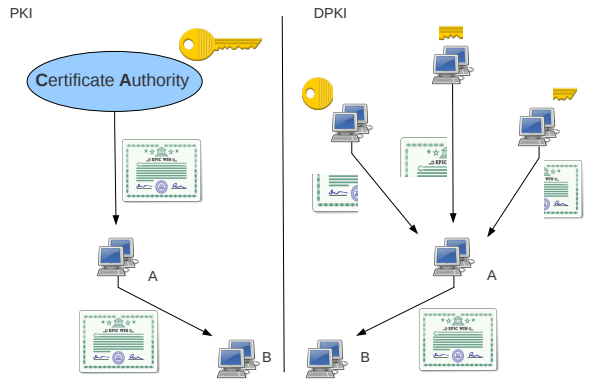

Figure 1: Left: A is issued a certificate by the CA and authenticates itself to B. Right: A is issued partial certificates and assembles the full fletched certificate to authenticate itself to B.

6. PROPOSED DISTRIBUTED P2P PKI

First we take a look at how the distributed PKI is bootstrapped. This includes initial key distribution and the basic layout of the network. Furthermore we explain how the PKI can be maintained by defining a set of operations on a structured P2P network like *Kademlia* as proposed by [3]. In theory, however, all operations discussed in the following can also be implemented for unstructured P2P networks with little extra work [3].

The idea behind this approach is that every node in the network knows only part of the private key $S = (e, m)$ that is used for certification. However, in the beginning this secret has to be generated. It is important that also in this generation phase no peer knows e completely at any time. To achieve this an efficient distributed algorithm already exists for RSA as proposed by [1]. After the generation of the private key S any peer in the network only knows the modulo **m** and $e_i | i \in \{1...n\}$ where n is the number of unique key parts or *shares*. Here it makes sense to introduce the concept of *Sharing Groups*. A Sharing Group consists of nodes that share the same information: Any peer in the Sharing Group i (SG_i) knows only the share e_i. This aggregation of peers into groups has the advantage that e_i is not instantly lost if a peer that knows this part of the secret leaves the network. Every share has a unique *shareId*. At the same time every peer in the network is assigned a unique *peerId*. The *shareId* is a prefix within the peerId in the form $peerId = shareId*$. Consequently when given a *shareId* it can be instantly determined if a peer knows this share (or at least part of it) by looking at its *peerId*. This feature combined with a structured P2P network makes it easy to find required shares that are needed to complete a certification process. Our approach for distributed certification is presented in the following.

6.1 Certification scheme

To perform an actual certification all Sharing Groups have to collaborate. Only if at least one peer of every Sharing Group takes part in the certification process it is possible to generate a valid certificate. An example certification process can look like this:
We assume there are three Sharing Groups ($SG_{shareId}$) in the network. Those are SG_0, SG_{10} and SG_{11}. An exemplary walkthrough is illustrated in 2 and the stages will be explained one by one in the following.

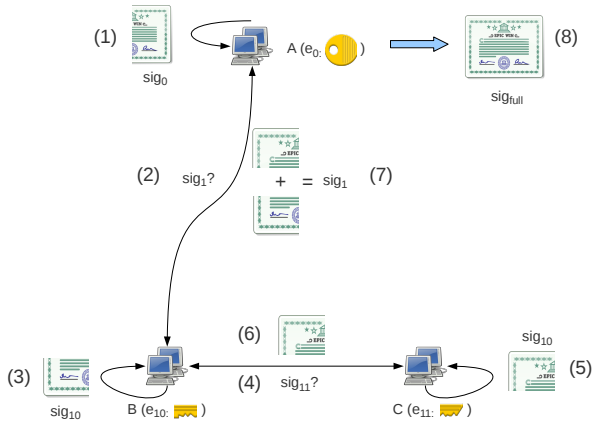

Figure 2: The distributed certification process.

The peer A, which is part of Sharing Group 0 (SG_0), wants to sign a certificate **cert**. Since A knows the share e_0 it can compute the partial signature sig_0 for the certificate **(1)**. The calculation of the partial signature looks like this:

$$sig_0 = cert^{e_0} \tag{2}$$

(Note: Generally when creating a signature a one-way hash function like SHA-1 is applied on the data and the resulting hash is signed. In our example, though, this step is omitted to keep it simple.)

As explained previously when using RSA the full signature can be calculated by multiplying all partial signatures. To generate the full signature $sig_{full} = sig_0 * sig_{10} * sig_{11}$ A requires the help of the other two Sharing Groups. A now asks a peer B that is in SG_{10} for the partial signature sig_1 **(2)**. This also requires A to send the certificate to B. B on the other hand does not know sig_1 completely but only sig_{10} **(3)**. Fortunately it holds that $sig_1 = sig_{10} * sig_{11}$ so B can

ask **(4)** any peer C in SG_{11} for the partial signature sig_{11} **(5),(6)** to calculate sig_1. After B sent sig_1 to A **(7)** the certification process can finish. A only needs to calculate: $sig_{full} = sig_0 * sig_1$. By using this signature A can complete the certification process **(8)**.

6.2 Maintenance operations

Peer-to-peer networks can be very dynamic. This means that peers frequently leave or join the network. Hence it is essential that the partial secrets distributed across the peers are redundantly available. Consequently there are multiple peers that know the same share of the public key S. Those peers are put into *Sharing Groups*. As long as there is at least one peer in each Sharing Group the network's public key S is safe. Before the network can be set up, however, it is necessary to settle for a maximum (max_{SG}) and minimum number (min_{SG}) of members in a single Sharing Group. As discussed above to successfully sign a certificate at least one peer of every Sharing Groups has to participate in the process. This implies that the ratio r of required peers depends on the maximum and minimum size of the Sharing Groups. In fact it holds that:

$$\frac{1}{max_{SG}} < r < \frac{1}{min_{SG}} \tag{3}$$

However, if all peers of a Single Sharing Group leave, the network's private key S is lost and cannot be recovered because the other peers, by definition, do not know S completely. To counter this a few maintenance operations are proposed in the following sections.

6.2.1 Split

If the size of a Sharing Group exceeds max_{SG} peers it should *split*. Without a split the ratio **r** would no longer be within its boundaries (see equation 3). This, however, is not the only reason why keeping the number of Sharing Groups high is a good idea. It also minimises the probability of a certain attack that can occur when there are only a few Sharing Groups. A detailed explanation of the attack is discussed later in section 7.

When a split occurs in a Sharing Group SG_i, the secret all peers in this group know (e_i) is split into two parts: $e_{i0} = \Delta$ and $e_{i1} = e_i - \Delta$ with Δ being a random number. Respectively Peers split into the new Sharing Groups SG_{i0} and SG_{i1}. A peer can determine by itself in which Sharing Group it splits because $peerId = shareId* = i*$ must still hold for i1 as well as i0. Consequently the peerId of a peer determines the Sharing Group it will belong to after a split. The split operation is illustrated in figure 3.

6.2.2 Refresh

The split operation has some drawbacks: Both Sharing Groups that result from a split know the original share e_i. This means that both Sharing Groups can calculate the other Group's share. For example SG_{i0} can calculate $e_{i1} = e_i - e_{i0}$. That is obviously a security flaw because no Sharing Group should know the share of any other Sharing Group.

To achieve this, [3] propose the *refresh* operation: After a split the Sharing Group selects a random other Sharing Group and they swap share information. By doing so the resulting share is no longer calculable by the other peers that split into the opposite Sharing Group. The refresh process is outlined in figure 4.

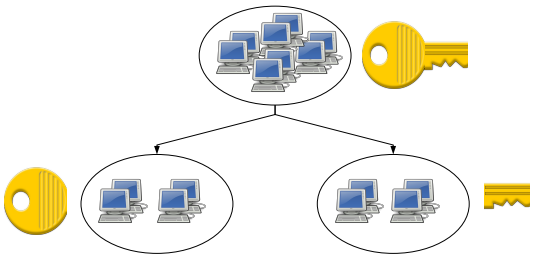

Figure 3: The split operation divides the secret across the new Sharing Groups. Key parts are representing the secrets.

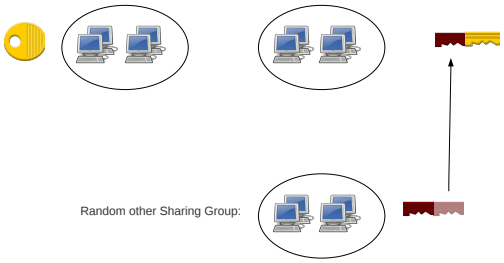

Figure 4: A Sharing Group performing the refresh operation after a split.

6.2.3 Merge

To ensure that the secret $S = (e, m)$ is not lost, it is important that no Sharing Group dissolves because all of its peers disconnect. Hence if the ratio of peers inside a Sharing Group SG_{i0} drops below $\frac{1}{max_{SG}}$ it simply merges with another Sharing Group SG_{i1}. After the merge, the Sharing Group SG_i consisting of all the peers that were previously part of SG_{i0} or SG_{i1} is created. The share all the peers in SG_i know is then simply calculated like this: $e_i = e_{i0} + e_{i1}$ The operation is illustrated in figure 5.

6.2.4 The Sharing Trees

The above operations work well in theory. However, they rely heavily on byzantine agreements. For instance peers always need to monitor their Sharing Group and all of them have to agree to split at some point. The same problem arises for the merge operation, of course. Also when the refresh operation is executed all peers of a Sharing Group

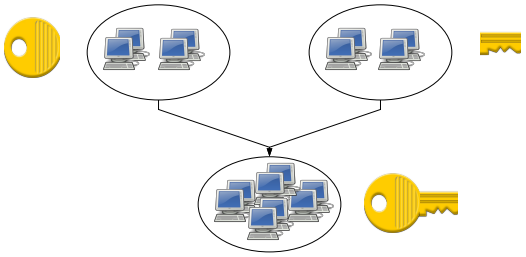

Figure 5: Two Sharing Groups merging together.

have to agree on a random value Δ and a random other Sharing Group. Those agreements are infeasible to fulfil in a practical implementation.

To solve this issue [3] propose the use of *Sharing trees*. Every Sharing Group is associated with such a Sharing tree. It defines for instance how the secret e is divided if a split occurs. This makes it possible for peers to split at different times because they will all split into the same Sharing Group and calculate their new secret in the same way. The secret e can be seen as the root of the main Sharing tree as illustrated in figure 6. If a peer decides to perform a refresh operation

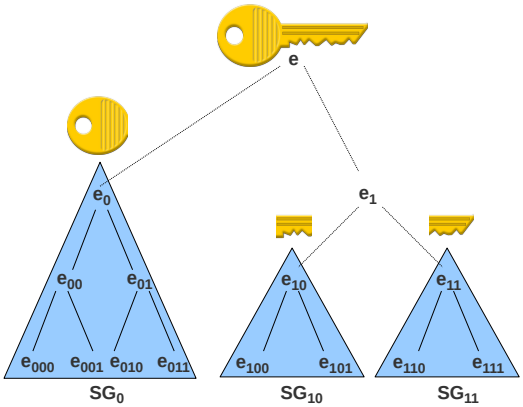

Figure 6: Here three sharing trees are highlighted of the three Sharing Groups SG_0, SG_{10} and SG_{11}.

it can select a random other Sharing Group and the random value Δ that will be exchanged. Then it tells all members of its Sharing Group to perform the refresh with its chosen variables. All peers in both Sharing Groups are able to update their Sharing tree correctly.

7. VULNERABILITIES AND DEFENCE

Just like with any system this proposed approach for a distributed PKI is not perfect. Two straightforward attack vectors are presented in the following along with probabilities of success and defensive measures.

7.1 Attacker in each Sharing Group

The first obvious issue to discuss here is: What happens if collaborating attackers manage to infiltrate every Sharing Group. This scenario would require **n** attackers sitting in **n** different Sharing Groups where **n** is the current number of Sharing Groups.

In this case the attackers can combine their knowledge and assemble the network secret $S = (e, m)$ resulting in a compromised PKI. Generally it can be said that the more Sharing Groups there are in the network the less probable this attack becomes if newly joining peers are always assigned to random Sharing Groups. Say the probability that a peer is an attacker is k_a. Now the amount of members of a Sharing Group i is g_i. The probability that all members of a Sharing Group i are *not* attackers is $(1 - k_a)^{g_i}$. Consequently the probability that there is in fact at least one attacker in SG_i is $1 - (1 - k_a)^{g_i} < 1$. Now we can also calculate the probability that there is *at least one* attacker in each Sharing Group:

$$P_{att} = \prod_{i=1}^{\#SG} 1 - (1 - k_a)^{g_i} \quad (4)$$

This equation proves that the probability of this attack (P_{att}) decreases if the number of Sharing Groups (#SG) increases. This is the most important reason why the *split* operation exists. By applying this operation it is possible to ensure that a reasonable high number of Sharing Groups exist. A visual representation of such an attack can be seen in figure 7.

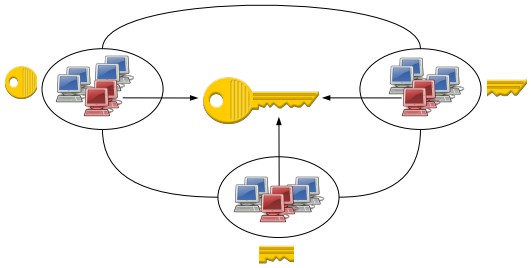

Figure 7: In this illustration three attackers from three different Sharing Groups participate in the retrieval of the secret key.

7.2 Misbehaving Sharing Group

Another possible attack consists of attackers "taking over" a single Sharing Group. This attack does not expose the network's private key S like the one discussed previously, but if a certificate needs to be signed the compromised Sharing Group can block the process by not answering to any requests. This attack, if successful, also makes the PKI unusable. Figure 8 illustrates the attack.

We can now look at the probability of this attack: Let k_a be again the probability that a peer is evil and g_i is the number of peers in Sharing Group i. The probability that

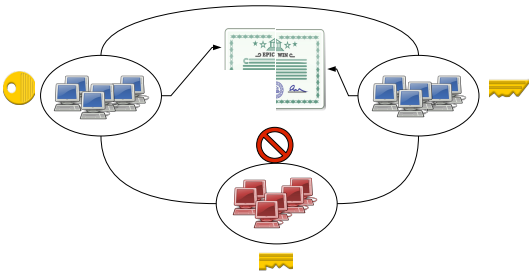

Figure 8: In this illustration two of three Sharing Groups participate in the certification process. The Group on the bottom, however, denies the final part.

there are only attackers in SG i is: ${k_a}^{g_i}$. Accordingly the probability that there is at least one peer in the SG that is not an attacker is: $1 - {k_a}^{g_i}$. Consequently the probability that there is at least one Sharing Group containing only attackers is:

$$P_{att} = 1 - \prod_{i=1}^{\#SG} 1 - {k_a}^{g_i} \tag{5}$$

Paradoxically every *split* operation theoretically raises the probability of this attack, because the less peers there are in a Sharing Group the more likely it is that all of those peers are attackers.

In conclusion it is obvious that the size limits of the Sharing Groups determine the success probabilities of those two attacks. Unfortunately minimising the likelihood of success of one attack increases the probability of success for the other. In other words the Sharing Group size describes a trade-off between those two issues and should be chosen wisely.

8. CONCLUSION

We presented an approach that can be used to distribute a public key infrastructure in a P2P network. It allows to put away with a central Certification Authority by distributing the certification process. Future work can be focused on making the system more robust against attacks. Also, as with any PKI implementation, certificate revocation is still an issue. The creation and distribution of the revocation lists (CRLs) would have to be distributed just like the decision process that determines what certificates are to be revoked and for what reason. An advantage in terms of security and availability of CRLs is that denial of service attacks against the PKI could be avoided. If the CRLs are properly distributed across a large number of peers the probability that a certification process fails because of an unavailable CRL could be minimised.

9. REFERENCES

[1] D. Boneh and M. Franklin. Efficient generation of shared rsa keys. *J. ACM*, 48(4):702–722, 2001.

[2] Y. Desmedt and R. Holloway. Some recent research aspects of threshold cryptography. In *In Proc. of the 1st Intl. Information Security Workshop*, pages 158–173. Springer-Verlag, 1997.

[3] F. Lesueur, L. Mé, and V. V. T. Tong. An efficient distributed pki for structured p2p networks. In H. Schulzrinne, K. Aberer, and A. Datta, editors, *Peer-to-Peer Computing*, pages 1–10. IEEE, 2009.

[4] R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, pages 552–565, London, UK, 2001. Springer-Verlag.

[5] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[6] L. Zhou, F. B. Schneider, and R. V. Renesse. Coca: A secure distributed online certification authority. *ACM Transactions on Computer Systems*, 20:329–368.