Universal Journal of Communications and Network 3(2): 51-56, 2015
DOI: 10.13189/ujcn.2015.030203 http://www.hrpub.org

A New Decentralized Cryptographic Access Control Solution for Smart-phones

Ernst Piller[1,*], **Fernando Moya de Rivas**[2]

[1]Institute of IT Security Research, University of Applied Sciences St. Pölten, Matthias Corvinus-Straße 15, 3100 St. Pölten, Austria

[2]Department of Communications and Information Technology, Polytechnique University of Cartagena, Pza. del Cronista Isidoro Valverde, Edif. La Milagrosa,30202 Cartagena, Spain

Abstract As the wireless technologies are becoming more and more popular and the number of cloud providers increases due to the growing demand of services and new emerging concepts like “Cloud computing”, security concerns are gaining a leading role. Data storage follows deprecated systems which make the cloud more vulnerable to data leakage. Besides, commmon cellphones have evolved to small pocket computers known as smart-phones, running their own Operating System. These devices are suffering every year an increment of cyber-attacks owing to weaknesses regarding the Operating System policy and the fact of using wireless standards such as GSM, UMTS, LTE, Wifi or Bluetooth. Over this document, we propose a new access control solution based on a cryptographic system managed by the end-user to add extra security to one’s personal information on the cloud. It is orientated to smart-phones and takes into account the computational limitations these devices may have.

Keywords Cryptographic Access Control, Rabin Cryptosystem, Shamir Secret Sharing Scheme

1 Introduction

In a world with a growing demand of services, where concepts like the Internet of Things (IoT) or Big Data are now a reality, and considering the rise of the global Internet connectivity as well as its speed average, the current global thinking has tended to move most part of resources to the cloud (Cloud Computing), filling the net with loads of data and increasing the number of cloud storage providers. In addition, the increment of smart-devices and wireless technology in general has caused people to be nearly permanently connected, sending constantly information to the Internet. As a consequence, end-users are highly interested in keeping their personal data securely on the cloud[8] instead of a physical device such as the hard disk or a pen-drive.

This issue has provoked a rise of net security concerns:

- Is our personal data safe?
- Who has access to this data?
- How can we assure its integrity?
- How to safely share this data?

Researches about this topic have discovered some weaknesses on most cloud storage providers, even the major ones, all of them concerning the way our files are shared. So as to put the reader into a context, there are usually two ways these servers provide for file-sharing: email and URL. The first option does not control which account the user gets logged in with. On the other hand, the second option does not assure that this URL is not gonna be widely distributed without the owner’s permission. Furthermore, it does not even always follows the HTTP protocol.

Therefore, while the file transference to the cloud is secure and commonly uses the TLS/SSL protocol, we come across two different problems. Firstly, the way in which these files are shared is not completely efficient. Secondly, the way they are stored is deprecated. It does not usually follow any cryptographic method or, in case it does, it is not possible for the end-user to use his own cryptographic key, that means, decentralize the process. As a result, we cannot be absolutely sure the administrator of the server has not accessed or even modified my personal data.

Over this document, we will explain a valid solution to cope with these security problems. We will call “owner” to that end-user who is in charge of the key management, that is, the administrator of the “sharing group”, the master. Likewise, we will name “partners”to those end-users who are part of the sharing group, but do not manage it.

2 ACL vs CAC

Regarding data storage, there are mainly two distinct solutions globally implemented:

- Access Control Lists.
- Cryptographic Access Control[6].

The former consists in a list of permissions which specifies which users are granted access and what kind of operations they are allowed to perform. Traditionally these operations are three at least: read, write and execute. ACLs are centralized which means there is a third party in charge of given these permissions. This is the main system found in current cloud storage providers.

ACLs can be thought of as a building. If someone wants to gain access to it, he must go to the entry phone, get identified and say which his intentions are. The system is centralized and it might keep a record containing all people accesses, timings, attendance. The problem is the information stored using this method is not encrypted and can be easily modified: a trusted central component is needed, not being suited for cloud storage providers. The content may have been even modified by the administrator of the building or someone who entered it by mistake, not being possible then to make sure of its integrity, authenticity and confidentiality.

The latter refers to that solution where the information is encrypted by its owner and decrypted for those who want to get access to this information, implying the necessity of a key management: generation, exchange, storage, use and replacement of cryptographic keys.

Depending on how this key management is performed, it is said the cryptographic access control is centralized or decentralized. On one hand, the key management can be done by the owner of the information. In this case, the key management is decentralized and has the advantage that the key is only known by the owner and not by a third party. Hence, it can be assured the information is authentic, has not been modified and no unauthorized person has accessed it. The problem appears when storing the key: there is no backup and in case the user loses its device or it gets broken, his data is lost too.

On the other hand, the key management can be performed by the cloud and in this case it is said it is centralized. Thus, the cryptographic key is always negotiated with the cloud, which is also in charge of its distribution and storage, having this way a backup. Nevertheless, the administrator or person in charge of the server could access at any time the data stored on it as the key has been agreed with the server.

In any case, if choosing a cryptographic access control, there is no impediment for a person who knows the key to have total access to the information as there is no difference between accesses: in this case, someone at the entrance of the building should get identified to get in it, being only capable of accessing the information he has the key for.

Eventually and so as to gather the advantages of both systems, the best solution is a combination of both approaches, where ACLs are used to grant access to some information which is encrypted, being safer that the key management is decentralized. Therefore, integrity, authenticity and confidentiality are guaranteed as well as restraining permissions are used.

3 Solution Adopted

The solution considered by our side is a decentralized cryptographic access control system, implemented for smartphones[7]. The choice of this platform meets the demands of a great deal of users worried by the rise of cyber-attacks over these devices[5]. Owing to their lower performance, issues such as memory capacity or computing speed are highly relevant and must be taken into account.

For being a decentralized solution, the owner must be the one who takes over the key management, including the key storage. Our approach consists in a system where cryptographic keys are related to each other, so that only one of them (the most recent one) must be stored since it would be enough for a user to calculate and obtain the previous versions from it. This is performed by a modular operation with a "public number" known by all the people a user wants to share his information with. At the same time, only the user who "starts" encrypting first is capable of computing the new key versions as he is the only one who has the means to do it: another modular operation regarding two "private numbers", whose multiplication outcome is the "public number". With the appropriate bit length, these private numbers are impossible to be achieved out of the public one.

So as to put the reader into a context, let's picture A wants to share some information with B and C. He generates a key and starts encrypting and sharing the ciphertext with B and C. Thus, another user called D cannot access this shared information as he does not know the key in use.

Now, let's figure A wants to share his information with C no longer. The limitations regarding memory capacity and computing speed prevent us from implementing a system where A had to download all his shared data, decrypt it, generate a new key, re-encrypt the data, re-upload it and share again the key with B. Instead, A could just generate a new key and start encrypting the new files with it, avoiding thus spending time in the process of re-encrypting the old data since C had already looked into the files, after all.

However, the old files would be encrypted with a different key so A and B should save both keys into the memory (again, a problem regarding capacity and, as we will see, a storage obstacle for a backup). Besides, if A wanted to share with a new user called E, he would have to distribute both keys instead of one, making the process more complicated. Hence, it would be desirable that the first key generated as well as the prospective ones had some sort of relationship between each other so that only one of them had to be stored, with the only addition of its version, that is, the position the current key occupy within the timeline. Therefore, E would receive the most recent key with which he could calculate the old versions and have access to every shared content, no matter they key that was used to encrypt them.

All in all, in terms of sharing information, our solution means some people who may have access to some data could lose their grant to new files (and/or subsequently modified ones, but not to old non-modified ones) if the key manager calculates a new key and does not distribute it or it does, but excludes some of them. It also means a new partner would only need the last key in use since he would be capable of calculating all its previous versions.

Hence, our concerns about speed and capacity in smartphones are fulfilled:

- Choosing the right algorithms will guarantee the process is performed fast enough, not blocking other processes that might be running on the device.

- The key management performance will permit to store data on the cloud and avoid downloading all the files to re-encrypt them in case a new file with different access grants (meaning someone is not given the key for this new share) has to be done.

- Only one key will be needed to be stored, which is the last version, since all the previous ones will be able to be computed by means of current key.

For being a decentralized solution, we have the additional problem of the key storage: presuming the administrator and owner of the key and the private numbers lost his phone or this last one was destroyed, there would be no way to continue with the secret sharing or at least to generate new keys. What is more: what if the owner is the only person in the group? His information will be lost forever. Summarizing, the point is the owner has no backup for these vital information (key and both private numbers).

One solution might be not to create any backup: in case the phone is lost, CACS key (key + private numbers) is lost too. Another one may be to store CACS key on one server and thereby have a backup on the cloud. Nevertheless, that is precisely what it was intended to avoid from the beginning: the server administrator having the key. CACS key could be even split into several parts and stored in different servers. It could be even mixed byte by byte following some algorithm. However, it will never stop being plaintext so, all in all, part of CACS key could be known, making easier to guess it since if one of these shares is sniffed, the sniffer will have part of the information. On the other hand, encrypting CACS key with a different key is not a solution either since we are in the same situation: what to store the key?.

Fortunately, it exists a good solution based on one of the last ideas exposed. CACS key could be split in several shares but following an threshold algorithm called "Shamir Secret Sharing"[14] by which the shares are securely protected and, likewise, one sniffer could even get no information out of several shares, let alone one of them. These shares could be, afterwards, uploaded to different servers and different parts of the cloud to make it securer (we do not want all of them stored on the same server). Besides, the reader must be aware that the transmissions client-server usually follow the TLS/SSL protocols, so even to sniff just one of these shares would be highly complicated for anyone.

4 Key Management

The purpose of this section is to describe the algorithms implemented and used for the key management: generation, exchange, storage, use and replacement of cryptographic keys.

The variables x and y will symbolize different versions of the key used by the system, p and q are the secret numbers generated by the "owner" and n is the public number, where $n = p \cdot q$

4.1 Generation

CACS solution is based on three "magic" numbers: a key and two prime numbers p and q. The first one must be shared with the persons somebody wants to securely share data with. Nonetheless, the two last ones are known just by the owner and only the outcome of their multiplication, n, is shared so as to compute the key replacement, that is, compute previous versions of the key in case they exist.

Hence, these three numbers must be generated, attending to some conditions, though. Firstly, p and q are computed, being verified they both are prime and fulfill:

$$p \equiv q \pmod 4 = 3$$

These conditions are to be satisfied for the Rabin Cryptosystem to perform valid results (see Replacement section). Then, another condition is required of the key: a positive Legendre and Jacobi Symbol regarding both p and q.

$$\left(\frac{y}{p}\right) = \begin{cases} 1 & \text{if } y \text{ is a quadratic residue modulo } p \\ -1 & \text{if } y \text{ is a quadratic non-residue modulo } p \\ 0 & \text{if } y \equiv 0 \pmod p \end{cases}$$

Likewise, if p and q are prime, y will be their quadratic residue respectively as long as:

$$y^{\frac{p-1}{2}} \pmod p = 1$$
$$y^{\frac{q-1}{2}} \pmod p = 1$$

As well as the other conditions, this one has to do with the Rabin Cryptosystem

.

4.2 Replacement

As it was commented on Section 3, CACS works with keys which are related to each other. More specifically, these keys depend on each other so that somewhat only one of them is needed to calculate the rest of them. For that purpose, the public number n and the private ones p and q are needed to be part of the operations regarding this behavior.

The algorithms invoked are Montgomery Reduction Theorem and both Chinese Remainder Theorem and Rabin Cryptosystem[9]. The former allows modular arithmetic to be performed efficiently when the modulus is large, while the combination of the last ones performs the quadratic residue.

In terms of efficiency, Montgomery Reduction Theorem can be optimized by means of the following iterative calculus called "Bunimov method"[2][13]:

```
P = 0;
for (int i = 0; i < n; i++) {
        P = P + xi * Y;
        if (P >= M) P = P - M;
        Y = 2 * Y;
        if (Y <= M) Y = Y - M;
}
```

where, $X = \{x_n, \ldots, x_1, x_0\}$ and the operation computed is $P = X \cdot Y \pmod M$. x_i stands for the bit in the position i.

To calculate the quadratic residue is more complicated and it needs several steps. The four possible solutions of the equation are obtained by invoking the Chinese Remainder Theorem:

$$x_1 = b \cdot s \cdot p + a \cdot t \cdot q \pmod n$$
$$x_2 = b \cdot s \cdot p + (p - a) \cdot t \cdot q \pmod n = n - x_1$$
$$x_3 = (q - b) \cdot s \cdot p + (p - a) \cdot t \cdot q \pmod n$$
$$x_4 = (q - b) \cdot s \cdot p + a \cdot t \cdot q \pmod n = n - x_3$$

Here, we have several terms to calculate. This is one of the reasons this operation is harder. Both a and b are the result of calling Rabin Cryptosystem. This algorithm has been commonly used as an asymmetrical encryption algorithm[1], with the disadvantage of having four possible solutions. If the plaintext is intended to represent a text message, guessing is not difficult. However, if the plaintext is intended to represent a numerical value, this issue becomes a problem that must be resolved by some kind of disambiguation scheme such as special structures or padding.

Nonetheless, its use here is with the purpose of computing an operation so having four different outcomes does not affect the behavior of CACS as long as they are valid solutions (which is true if the restrictions are fulfilled, for the key, p and q; see 4.1) since only one of them is needed. Therefore, a and b can be computed as follows:

$$a = \sqrt{y} \pmod p \quad \rightarrow \quad a = y^{\frac{p+1}{4}} \pmod p$$

$$b = \sqrt{y} \pmod q \quad \rightarrow \quad b = y^{\frac{q+1}{4}} \pmod q$$

The last terms left are known as the coefficients of Bézout's identity, which can be obtained by means of the Extended Euclidean Algorithm (whose entries in this case are the secret numbers p and q). This computes the greatest common divisor (gcd) of two integers a and b as well as the previously mentioned coefficients. The Bézout's coefficients are two number s and t that fulfills:

$$p \cdot s + q \cdot t = gcd(p, q)$$

being p and q two integer numbers, with $p > q$. In order to calculate Bézout's coefficients, an iterative calculus must be applied which consists of computing a sequence of quotients u_k and remainders r_k, with

$$r_0 = p \qquad r_1 = q$$

The initial values for s and t are:

$$s_0 = 0 \qquad s_1 = 1$$

$$t_0 = 1 \qquad t_1 = 0$$

and the next values are calculated as follows:

$$s_{i+1} = s_{i-1} - u_i \cdot s_i$$

$$t_{i+1} = t_{i-1} - u_i \cdot t_i$$

The computation is finished when $r_k + 1 = 0$. Likewise, all terms needed for the Chinese Remainder formula are obtained and CACS is set to perform the key replacement (and calculus of previous keys).

4.3 Use and exchange

Regarding the key use and exchange, the procedure basically follows the common standards used worldwide, except some issues that we will discuss. For the key distribution, an asymmetric scheme is used wheras a symmetric one is used for the data encryption. Thus, our solution makes use of a hybrid key exchange (like PGP solution) with RSA[10] and AES[3] respectively. The encryption is performed using AES with "ECB-mode" for the file name and "CBC-mode" for the file content. Concerning this last one, a random initializacion vector (IV) is randomly generated and appended at the end of the ciphertext so that the decryption cipher knows the sequence to use.

Another issue is the AES key length. AES standard consider three different lengths: 128, 192 or 256 bits. For security reasons, Rabin Cryptosystem demands 1024-bits numbers so as to be robust enough and not easy to be hacked. As a consequence, a hash function (SHA-256[12]) is applied to get an appropriate length (256 bits in this case).

5 Storage

Regarding the key storage, Shamir Secret Sharing scheme is applied. In cryptography, a secret sharing scheme is a method for distributing a secret amongst a group of participants, each of which is allocated a share of the secret. The secret can only be reconstructed when the shares are combined together as individual shares are of no use on their own.

More formally, in a secret sharing scheme there is one dealer and n players. The dealer gives a secret to the players, but only when specific conditions are fulfilled. The dealer accomplishes this by giving each player a share in such a way that any group of k (for threshold) or more players can together reconstruct the secret but no group of less than k players can. Such a system is called a (k, n)-threshold scheme[11]. Particularly, Shamir Secret Sharing scheme make use of polynomial interpolation ("Lagrange interpolation") and this is the reason this scheme is so popular.

The essential idea of Adi Shamir's threshold scheme[14] is that 2 points are sufficient to define a line, 3 points are sufficient to define a parabola, 4 points to define a cubic curve and so forth. That is, it takes k points to define a polynomial of degree $k - 1$.

Suppose we want to use a (k, n) threshold scheme to share our secret S, without loss of generality assumed to be an element in a finite field F of size P, where $0 < k \leq n < P$, $S < P$ and P is a prime number.

Choose at random $k-1$ positive integers $a_1, a_2, \ldots, a_{k-1}$ with $a_i < P$, and let $a_0 = S$. Build the polynomial $y = f(x) = a_0+a_1x+a_2x^2+a_3x^3+\ldots+a_{k-1}x^{k-1}$. Let us construct any n points out of it, for instance set $i = 1, 2, \ldots, n$ to retrieve $(i, f(i) \mod p)$. Every participant is given a point (an integer input to the polynomial, and the corresponding integer output). Given any subset of k of these pairs, we can find the coefficients of the polynomial using interpolation. The secret is the constant term a_0.

In order to reconstruct the secret we will compute Lagrange basis polynomials:

$$l_0 = \left(\frac{x - x_1}{x_0 - x_1} \cdot \frac{x - x_2}{x_0 - x_2} \cdot \ldots \cdot \frac{x - x_{k_1}}{x_0 - x_{k_1}} \right) \pmod p$$

$$l_1 = \left(\frac{x - x_0}{x_1 - x_0} \cdot \frac{x - x_2}{x_1 - x_2} \cdot \ldots \cdot \frac{x - x_{k_1}}{x_1 - x_{k_1}}\right) \pmod p$$

$$\vdots$$

$$l_{k-1} = \left(\frac{x - x_0}{x_{k-1} - x_0} \cdot \ldots \cdot \frac{x - x_{k-2}}{x_{k-1} - x_{k-2}}\right) \pmod p$$

Therefore,

$$f(x) = \sum_{j=0}^{k-1} y_j \cdot l_j(x)$$

6 Comparison: Solutions on the market

Currently on the market, there are several tools based on decentralized access control systems. They split and, therefore, distinguish between storage and confidentiality responsibilities, hence emphasizing, once more, that a cloud storage provider should only be responsible for storing data and regulating user-level access. There are two main kinds of tools: container and file based encryption tools. The latter is the one in the matter at hand.

These file encryption tools usually work as a third-party for a great number clouds, such as Dropbox, Google Drive, Box or Microsoft One Drive. We will highlight two of them: Boxcryptor and TAVUU.

The first one defines themselves as an "easy-to-use encryption software optimized for the cloud", supporting a great deal of different cloud storage providers and OS platforms. The file encryption is performed symmetrically and the file key is added to the resulting bunch of bits after an asymmetrical encryption. It fulfills the "Confidentiality as a Service" paradigm[4] in which a multiple commutative layers of encryption scheme is used. This means these layers can be added and removed in an arbitrary order.

The second one is a peer-to-peer solution which avoids the use of a server, all in all a client-based software solution. As in the case above, the file encryption is symmetrical and it makes use of an asymmetric scheme for the key exchange. The foremost idea is this solution is a hierarchical system, where one user generates a key as "an entry point" key, which has a public and a private part and grants access to the parent directory. Keys for subfolders are discovered by traversing the file system, depending whether you are authorized. This authorization comes from a user, called owner or producer, who can also deauthorize other users, making them incapable of seeing new files or changes.

The flaw of this approach is its incapability for backup: as it is a non-server solution, one users password will never leave his end-point device (computer, mobile phone...). Thus, as a side-effect it is impossible to recover it in case of forgetting or damage. Our key management approach follows a modern and secure solution for the key storage, which offers a user-friendly experience based on a (k, n)-threshold scheme. The upset is the more shares one wants to split the key in, the more accounts and password one has to remember.

Besides, these two presented solutions are focused more on computers than smart-devices. The algorithms they use do not considered their computational and memory limitations and, thereby, are not optimized for the use of smart-phones or tablets. Our solution is more time-efficient in this sense and, by means of the Rabin-method and the idea of relating keys to each other, no re-encryption is needed after the owner decides to exclude one of his partners from the sharing group.

7 Implementation and Testing

This approach has been implemented and tested in an Android environment. We will now attend to some considerations about its implementation. The resulting application makes use of different folders for the synchronization in which we can find:

- Plain files.
- Encrypted files (for testing).
- Meta data needed for the synchronization (revision number on the server and timestamp in the smart-device).
- The CACS key, RSA pair of keys and both secret numbers, in case of the owner; the CACS key, RSA pair of keys and the public number, in case of the partner.

The App enables file and folder uploading, downloading, renaming and deletion. Client-server operations are carried out immediately while server-client ones are performed with a 10 seconds latency. All these operations need a multithreading system.

It has been tested exhaustively by a random group of students within a period of one month, installing the App in their smart-phones. During this time, several development bugs were corrected to improve its operation and its simple user-friendly interface. The following considerations arose within this test:

Process	**Milliseconds**
Key and secret numbers generation	10 GB
Key storage	10 GB
Key recovery	100 MB

Table 1. CACS operation results. Except the last measure, the rest depends on the speed of the prime number generation. The wide range on the first measure is due to the restrictions the three numbers must fulfilled between each other.

The rest of operations (key exchange, use and key substitution) were performed time-efficiently, barely consuming time. By these, it was deduced the algorithms CACS solution implements are suitable for smart-devices, considering the background idea, as well.

8 Conclusion

Conceivable bugs and weaknesses that might be on cloud storage providers are coped by our approach, adding a new grade of security to data storage. By making use of the Rabin Cryptosystem from an uncommon point of view, we are capable of setting up a entire cryptographic system which fulfills with the smart-phones computational limitations. We propose a decentralized solution where it is the end-user, and not any third party, who manages the key and we give a solution for the key storage, based on a (k, n)-threshold scheme.

Over the paper, we have commented one by one all the pieces which form part of the key management. This system has been implemented and tested on Android, but it is itself a solution not only valid por Android smart-phones, but general purpose. Finally, we have given several reasons for which we think cryptographic access control should be added to current cloud data storage.

REFERENCES

[1] Gilles Barthe, David Pointcheval, and Santiago Zanella Béguelin. Verified Security of Redundancy-free Encryption from Rabin and RSA. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 724–735, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1651-4. doi: 10.1145/2382196.2382272.

[2] Viktor Bunimov and Manfred Schimmler. Completely redundant modular exponentiation by operand changing. In *Proceedings of the 2005 International Conference on Computer Design, CDES 2005, Las Vegas, Nevada, USA, June 27-30, 2005*, pages 224–232, 2005.

[3] Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002. ISBN 3540425802.

[4] S. Fahl, M. Harbach, T. Muders, and M. Smith. Confidentiality as a service – usable security for the cloud. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*, pages 153–162, June 2012. doi: 10.1109/TrustCom.2012.112.

[5] Minzhe Guo, Prabir Bhattacharya, Ming Yang, Kai Qian, and Li Yang. Learning Mobile Security with Android Security Labware. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 675–680, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1868-6. doi: 10.1145/2445196.2445394.

[6] Anthony Harrington and Christian Jensen. Cryptographic Access Control in a Distributed File System. In *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies*, SACMAT '03, pages 158–165, New York, NY, USA, 2003. ACM. ISBN 1-58113-681-1. doi: 10.1145/775412.775432.

[7] Takashi Matsunaka, Takayuki Warabino, and Yoji Kishi. Secure Data Sharing in Mobile Environments. In *Proceedings of the The Ninth International Conference on Mobile Data Management*, MDM '08, pages 57–64, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3154-0. doi: 10.1109/MDM.2008.32.

[8] Machigar Ongtang, Kevin Butler, and Patrick McDaniel. Porscha: Policy Oriented Secure Content Handling in Android. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 221–230, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0133-6. doi: 10.1145/1920261.1920295.

[9] M. O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical report, Cambridge, MA, USA, 1979.

[10] Dorothy Elizabeth Robling Denning. Rivest-Shamir-Adleman (RSA) Scheme. In *Cryptography and Data Security*, pages 104–109. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1982. ISBN 0-201-10150-5.

[11] Dorothy Elizabeth Robling Denning. Threshold Schemes. In *Cryptography and Data Security*, page 179. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1982. ISBN 0-201-10150-5.

[12] Somitra Kumar Sanadhya and Palash Sarkar. A New Hash Family Obtained by Modifying the SHA-2 Family. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ASIACCS '09, pages 353–363, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-394-5. doi: 10.1145/1533057.1533103.

[13] Manfred Schimmler and Viktor Bunimov. Fast modular multiplication by operand changing. In *International Conference on Information Technology: Coding and Computing (ITCC'04), Volume 2, April 5-7, 2004, Las Vegas, Nevada, USA*, pages 518–524, 2004. doi: 10.1109/ITCC.2004.1286707.

[14] Adi Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, November 1979. ISSN 0001-0782. doi: 10.1145/359168.359176.