Vision: What If They All Die? Crypto Requirements For Key People

Chan Nam Ngo
University of Trento
Trento, Italy
channam.ngo@unitn.it

Daniele Friolo
Sapienza University of Rome
Rome, Italy
friolo@di.uniroma1.it

Fabio Massacci
University of Trento
Trento, Italy
fabio.massacci@unitn.it

Daniele Venturi
Sapienza University of Rome
Rome, Italy
venturi@di.uniroma1.it

Ettore Battaiola
Cassa Centrale Banca
Trento, Italy
ettore.battaiola@cassacentrale.it

***Abstract*—The question above seems absurd but it is what a Bank has to ask to its suppliers to meet the European Central Bank (ECB) regulations on the continuity of critical business functions. The bank has no intention of mingling in the daily work of the supplier (that's the whole purpose of outsourcing). Nor the supplier has any intention to make available to the bank the keys of its kingdom (it is actually forbidden to do so by the very same regulations). We need a way to do so only when the hearts of the key people stop beating. In this paper, we discuss whether recent advances in cryptography (secret sharing and MPC, time-lock puzzles, etc.) can replace the classical approach based on human redundancy.**

1. Introduction

Encryption [6], [18] is among the most popular mechanisms for data security. Indeed, banking systems use gazillions of encryptions to keep data confidential, at rest or in transit. Although not usually explicitly mentioned, decryption is necessary for this mechanism to work as intended: somewhere, somehow, somebody needs to supply passwords which nobody wrote down, or to pass a live biometric test binding the decryption key [4].

For large organizations, there might always be somebody *alive*, but for smaller organizations things may be different. A small organization might specialize in a particular service that is critical to a bank (or a set of banks). Its IT staff might run into an handful as its core business is not the IT, it is just the particular service that is a small cog in an overall machine churning credit (a critical, heavily regulated, business function).

One of such examples is the administrative management of small and medium enterprises (SME)'s applications to the Guarantee Fund for the Italian Ministry of Finance (other European countries have a similar scheme). The application form is complicated and must change at the whims of the Ministry. Getting the form submitted in due order is a critical requirement for the SME credit worthiness at the Bank, albeit, from the perspective of credit scoring, this is just a couple of bits in the function (submitted, approved). Hence, companies specialize in this application process to allow banks to concentrate on the core business. The core of the business is not IT, it is the knowledge of how to fill the forms. IT security is important (as this is also regulated by the ECB), but staff members might be few as otherwise the whole business would be unprofitable.

In this scenario a legitimate question the bank must ask to its supplier is precisely *what happens if they all die*: no single system admin who knows a password or owns the necessary biometrics is left.

As an example, after the tragic 9/11 incident, the first thing that came to the mind of the chief executive of Cantor Fitzgerald, one of the world's largest financial-services firms, was passwords.[1]

The formal phrasing is of course different from bank to bank, from supplier to supplier, and most likely not in such extreme terms,[2] but the essence is the same: what do you do if the critical employees holding the key to the kingdom are no longer there?

In a less tragic case, the importance of having such a plan has become well known to the general public with the case of Terry Childs. Terry Childs was the network administrator of the city of San Francisco. In July 2008, he refused to hand over the passwords that grant administrative control to the city's FiberWAN network, because his supervisors were not qualified to have access to the passwords.[3] Terry Childs was eventually sentenced to 4 years and had to hand over the passwords to the city's mayor. Even though the city's network operated normally during the 12 days of the incident, the city had to pay an expensive bill of $900,000 on trying (and failing) to regain control of its own network. Fortunately, in this case, Terry Childs was not dead, therefore the passwords could still be recovered (only by law enforcement). In our worst case, the city would have spent much more for the reconfiguration of its network. The European Union Agency

1. https://www.nytimes.com/2014/11/19/magazine/the-secret-life-of-passwords.html.

2. The usual form for the ECB guidelines is "Essential employees are unavailable". Yet, we know of one case in which it was exactly formulated in the way we mention.

3. https://www.cio.com/article/2416252/network-admin-terry-childs-gets-4-year-sentence.html.

for Cybersecurity (ENISA) provides no guideline for this scenario, while the National Institute of Standards and Technology (NIST, in the US) only provides somewhat general guidelines for Key Recovery Systems [1] whose purpose is to backup and recover encryption keys.

Orgaizational solutions are the current mechanisms of choice (separation of duties, redundancies etc.) but we wonder if some security protocols could be set-up to avoid such measures and provide a purely technical solution. We can of course create *escrow systems*, e.g. the passwords are deposited at a regulator but that's a violation of security as we have no way to make sure the system is not abused.

State machine replication [3], [11], where the bank outsources the information processing to more than one IT supplier, is another trivial solution. This is a cryptographically sensible solution, except that it requires *at least* three suppliers (instead of one) and fails the whole point of saving costs in oursourcing.

The bank could also share the data among its IT suppliers, e.g. using *secret sharing* [21], and later run a *secure multiparty computation* (MPC) [5] protocol between the bank and the suppliers for the procedure to get some processed information from the repositories of suppliers.[4]

However, there will be a need of additional computational power together with larger storage requirements, developing the application to manage the new procedures, etc.All these are hardly worthy to manage the SMEs formal applications to the Ministry of Finance Guarantee Fund.

One could also develop an automatic recovery procedure, where the system admins must periodically perform an action, otherwise the recovery procedure kicks in. We call this a heart-beat puzzle.[5] In this case the time window becomes a security parameter, similarly to what happens with time-lock puzzles, i.e. encryptions that can be broken after a certain period of time.

Paper Organization. The rest of the paper is organized as follows. In §2, we provide more details to the "what if they all die" scenario. There, we also outline the considered adversarial model and acceptable trust assumptions. To make the paper self-contained, we review a few necessary cryptographic primitives in §3, followed by possible solutions to our question using those schemes in §4. Yet, in §5, we argue that those solutions might not work (in their current form). Thus, in §6, we envisage a new cryptographic primitive, which we call the heart-beat puzzle, to tackle the mentioned problem. Finally, we conclude the paper in §7.

2. System Model: What If They All Die?

The scenario we consider is the outsourcing of a small cog in a larger machine that is mission critical for an institution.

- Absolving the cog functions requires specialized knowledge in which the IT function is purely supporting to the process.
- The cog is actually small in the overall (critical) process, so, on the one hand, it is "critical" according to the ECB definition, and on the other hand it is not so critical from the institution's perspective to the point of being necessarily internalized.
- The cog might have associated a reasonably small IT staff, so that a simple car incident involving the people traveling to a customer's meeting might wipe out the key people.[6][7]

The example we mention here is the administrative management of an SME's applications for the Guarantee Fund for the Italian Ministry of Finance. Upon successful approval of such application, the SME might qualify for a one-off guarantee by the Ministry for around 80% of the balance sheet. Such forms may change every year depending on the particular ministerial decree specifying the technical conditions, and must often supplied electronically. The information for the application must be digitally signed in appropriate ways, and a number of equally digitally signed documents must be supplied as annexes. Hence, a company will typically specialize in supporting this application process by keeping itself up to date with the regulation and the type and format of document necessary for successfully submitting the application and processing the responses. Such company might be small by itself.

The key in this process is that an SME would normally apply to the Guarantee Fund through the bank and not through the company specializing in such applications. Indeed, the whole process is just a cog in the overall process that the bank has to undergo in order to supply credit to the SME. So the form management company ends up being actually a critical supplier of the bank.

Therefore, the form management company must now show that they have a number of critical security functions and, among them, the ability to manage the disappearance of the critical IT employees. For the form management company, the duplication of the critical employees is obviously a financial burden. One cannot have IT employees holding the key of the kingdom paid a pittance as the chance that they would reveal some data or misuses some data is higher.

*System Model.*We consider two main entities: the *Bank* who owns the data and the *Suppliers* who perform non-critical functions for the bank. The bank outsources some (initial) data to the suppliers for them to start performing the required functionality. During the operations, the suppliers can produce and store intermediate data. Since the bank is only interested in the final result, those intermediate data might not be of the bank's interest. Therefore they are often only stored at the suppliers' storage. For security reasons, the suppliers must[8] encrypt their data. The encryption key is only known to the sysadmin of

4. Threshold Encryption [19], where t out of n admins *of the same IT supplier* can jointly encrypt/decrypt the data without revealing the key, is clearly not usable.

5. We can think of the "heart-beat" as the parties who hold the key "do something periodically".

6. Of course here we mean the key IT administrators. The sudden death of the CEO or the janitor might both create havoc within a company, but we ignore this aspect for the time being.

7. A trivial solution would be to forbid key people traveling together. Unfortunately this is only viable if the company is a big corporation with 1000 employees where one key person could be accompanied by other supporting staffs. If the company has only 10 people including the accountant and the technicians, we clearly have a problem with this setup.

8. Unless we use secret sharing and MPC, see §4.

the respective supplier. Clearly, the admin must be trusted in this settings.[9]For simplicity, this is also our only trust assumption. Additional requirements might change this trust assumption, but we do not consider them here. The problem arises when the sysadmin is unavailable. The bank then has the need to recover the key (and thus the data, including intermediate ones) to let another sysadmin or supplier take over the job.

Cryptography could provide a one-stop-shop solution: the (expensive) system administrators could present a "heart-beat" (the parties who hold the key "do something periodically"). If they fail to do it within a certain time window then a recovery procedure kicks in. At this point the key can be made available to a senior executive (e.g. CEO/Chair) who could pass it on to the next expensive sys admin who is taking over. The latter may immediately change the password and restore the heart-beat procedure. The time window becomes a security parameter.

*Adversarial Model.*We can model the main entities in the adversarial model as the Bank (B) and the suppliers (SPs). B can be modeled as a semi-honest adversary that engages in some protocol with the SPs when a particular event where B's interaction is needed occurs. A specific actor working for the SP, called sysadmin, is trusted. All the other actors working for the SPs can be considered semi-honest adversaries instead, meaning that they can have access to the database and read its content. The database therefore must be encrypted with a key known only to the sysadmin. The security goal here is to ensure that B and the external adversaries cannot learn the key as long as the sysadmin of the SP is operational. Being non-operational can be formalized by the fact that the sysadmin doesn't send an heart-beat after a fixed time-shift. In this case a key-recovering procedure can be applied by any other actor of the SP, therefore allowing to learn the key and decrypt the database.

3. Background: Cryptographic Schemes

To make the paper self-contained, in what follows we briefly review the candidate cryptographic techniques.

3.1. Secret Sharing and MPC

We start by reviewing secret sharing and MPC. We focus more on the former, as the latter would only be performed among the suppliers.

Intuitively, *secret sharing*, allows a party to share a secret s among n parties, in such a way that any t parties can recover the secret. On the other hand, any set of less than t parties cannot learn any information about the secret. A secret sharing scheme consists of two probabilistic polynomial time (PPT) algorithms (SS.Share, SS.Recover):

$\{\mathsf{s}_i\}_{i=1}^{n} \leftarrow \mathsf{SS.Share}(\mathsf{s}, t, n)$ takes as inputs the secret s, the threshold t and the number of parties n (with $t \leq n$) and outputs a share s_i for each party i.

$\mathsf{s} \leftarrow \mathsf{SS.Recover}(\{\mathsf{s}_j\}_{j\in\mathbf{T}}, t)$ takes as inputs the set of shares $\{\mathsf{s}_j\}_{j\in\mathbf{T}}$ where $\mathbf{T} = \{i : 1 \leq i \leq n\}$ and $\|\mathbf{T}\| \geq t$, and outputs the secret s.

9. Terry Childs can be technically considered as "honest but unavailable" sysadmin. A malicious admin can just delete the ciphertext or reveal the plaintext.

Secret sharing schemes must satisfy *Recoverability* and *Secrecy*.

Recoverability The recovery of the shared secret s is successful with overwhelming probability given at least t shares $\{\mathsf{s}_j\}_{j\in\mathbf{T}}$, where $\mathbf{T} = \{i : 1 \leq i \leq n\}$ and $\|\mathbf{T}\| \geq t$.

Secrecy Any set of at most $t-1$ shares $\{\mathsf{s}_j\}_{j\in\mathbf{T}}$, where $\mathbf{T} = \{i : 1 \leq i \leq n\}$ and $\|T\| < t$, leaks nothing about the shared secret.

Remark 1. Let us note that one can also detect malicious dealers in secret sharing [8], where an adversary sending incorrect shares can result in a protocol abort. This kind of secret sharing schemes require one round to share a secret and one round to recover it. The communication complexity is only $\mathcal{O}(n)$ for broadcasting the shares.

3.2. Time-Lock Puzzles

Roughly speaking, a time-lock puzzle [12], [13], [17] is a hard problem parameterized by a time-lock t, so that the best possible algorithm for solving the puzzle takes t steps and the solver cannot be parallelizable. This means that all attacks can only be performed by a single sequential machine, and the time τ that it takes to find the solution relies only on its speed.[10]

To be precise, a puzzle [2] is a tuple of algorithms (Puzzle.Gen, Puzzle.Sol):

$Z \leftarrow \mathsf{Puzzle.Gen}(t, s)$ takes as an input a parameter t and a solution $s \in \{0,1\}^{\lambda}$, where λ is the security parameter, and outputs a puzzle Z.

$s = \mathsf{Puzzle.Sol}(Z)$ is a deterministic algorithms taking as an input a puzzle Z and outputting a solution s.

Differently from the known crypto primitives, for the puzzles we consider the efficiency as part of the definition. The security properties are stated as follows:

Completeness For every security parameter λ, difficulty parameter t, solution $s \in \{0,1\}^{\lambda}$ and puzzle Z in the support of $\mathsf{Puzzle.Sol}(t, s)$, $\mathsf{Puzzle.Sol}(Z)$ always outputs the solution s.

Efficiency Some puzzle Z computed with $\mathsf{Puzzle.Gen}(s, t)$ for some solution s can be computed in $poly(\log t, \lambda)$ and the solution can be computed by $\mathsf{Puzzle.Sol}(Z)$ in $t \cdot poly(\lambda)$.

Security The parallel time required to solve a puzzle is proportional to the time it takes to solve the puzzle honestly (i.e. by using a sequential strategy), up to some fixed polynomial loss.

Remark 2. By the definition of efficiency, the solution for some puzzle takes more time to be computed by changing the difficulty parameter. A common instantiation of time-lock puzzles is using the RSA assumption [18].

4. Possible Solutions: Crypto To The Rescue

Escrow Systems. This is the most convenient solution, where the passwords are deposited at a regulator. The

10. To make τ constant, the parameter t, in facts, must be increased as faster chips are released.

Clipper Chip was an example of such escrow system.[11] Any crypto device integrating a Clipper Chip (e.g. a crypto phone) would give its cryptographic key to the government in escrow. A “trusted” government agency could therefore “claim authority” for intercepting and decrypting a particular communication transmitted through the mentioned device.

State Machine Replication. The bank could consider outsourcing the information processing to an IT supplier. The bank could send the business related data to its supplier,[12] and have the supplier process the data. The supplier then will return the result to the bank. In this case, each IT supplier has its own key and encrypted data. If one supplier dies, the others are still available to run the services.

Secret Sharing and MPC. It is possible to store the bank data in a redundant and encrypted form at a set of IT suppliers' data storage using secret sharing techniques, e.g. a bank could use Shamir's t-out-of-n secret sharing scheme (SS) [21] to generate cryptographic shares and store them at the IT suppliers' servers. Later, general MPC [5], [22] can be used to compute a function from the shares to process some information that is necessary to the bank (note that it is sufficient that the crypto computation is all performed by the suppliers, the bank only needs to reconstruct the output from the computed shares, and it is happy about that!). A bank, in order to avoid paying for expensive in-house servers and the overall operation/maintenance costs, could decide to outsource its data s as cryptographic shares $\mathsf{s}_1, \ldots, \mathsf{s}_n$ to some external servers owned by the bank's suppliers $S_1, \ldots, S_n$. In this way the servers store their corresponding banking data in an encrypted and redundant fashion.[13] Whenever the bank needs to retrieve or process some information, it starts some specific MPC protocol with the servers in order to evaluate a function $f(\{\mathsf{s}_i\})$ (which is mostly done by the suppliers, the bank only reconstructs outputs).Note that there is *no* encryption key to be stored and recovered in this case, as the data is stored as shares at the servers (of equal roles), which is sufficiently secure. When one supplier dies, the $n-1$ suppliers can still run the services (we only need $t < n$). However, the bank might need to add another supplier (and resharing the secret), but this is easily doable.

Time-Lock Puzzles. We can use time-lock puzzles to solve our problem by requiring that the system admin updates his current key k to some new key k', re-encrypts the data with new key, and publishes (e.g. posting it on a blockchain) a time-lock puzzle $Z \leftarrow \mathsf{Puzzle.Gen}(t, k')$ of the new key k' setting t such that the best known machine can solve the puzzle at least in some time τ. If the system admin stops to update the key, then it can be retrieved after a time τ by some other actor of the system (e.g., the bank's CEO or a regulator)by invoking $k' \leftarrow \mathsf{Puzzle.Sol}(Z)$.

11. https://www.cryptomuseum.com/crypto/usa/clipper.htm
12. Albeit the bank can still store some hashes of the data for later verification.
13. The suppliers could also outsource the data storage to another third party.

5. Limitations: Will These Solutions Work?

Key escrow is the most convenient alternative, but also the least secure one, since a Trusted Third Party (TTP) is required. Such a system introduces another point of failure as it is easy to be abused. Indeed, in presence of an escrow, abuses will eventually happen as the temptation is far too great: the escrow itself means access to the original system *as if* they all died. So in our scenario the bank employee holding the escrow might collude with the SME to warrant them an “application successful checkmark” that wasn't existing. Such a solution is clearly not preferable if we are dealing with bank data.

State machine replication is a cryptographically sensible solution, except that it requires *at least* three suppliers (instead of one) for State Machine Replication. As such, its security is as weak as the Key Escrow mechanism. The bank now has to trust at least three suppliers with its raw data. Moreover, the whole point of oursourcing was to save money and now this cost *at least* tripled. Further, the current regulations makes sure that the suppliers will have to manage redundancy but in this way IT redundacy was a supplier's problem. With the multiple servers, it has become a bank's problem.

In the case of *secret sharing and MPC*, the bank now has the burden of trusting at least t servers to be honest.Moreover, even if the banks are happy that the data-outsourcing servers do the job during normal operations with no action on their side, this solution could require heavy machinery and does not scale very well. First of all the cost is now multiplied by n which fails the point of oursourcing the process to save money. Further, in this way, the bank may still have to manage redundancy by itself (by regulations) and it still needs to buy more cryptographic hardware, manage the secret sharing process, etc.

Time-lock puzzles, at first sight, seem like a sound solution. Yet, they are not (in the current form). Suppose we have a time-lock puzzle which is locked until tomorrow (τ is 1 day). In this case, one may think to have the heartbeat arrive a few seconds before dawn (the admin updates the key, re-encrypt the database, and publishes the new time-lock puzzle) so we should get an extra day.

This is very true... if the bad guys try to break the new puzzle (what honest bad guys!). As the time for breaking the old puzzle has anyhow passed, the bad guys will actually have broken the old locked key.

The obtained key is old, yet useful. The attackers might also have access to the old encrypted data if they can still obtain the data of yesterday. Even in futures trading where everything should conclude within the day, such trading data still gives insights on traders' strategy and that could be hazardous [14]. Maintaining confidentiality in this scenario is also known as Forward Secrecy [9], [15]. Unfortunately, it cannot be inherently achieved if the data is static, meaning that the plaintext doesn't change over time and it's still considered sensitive. One could possibly solve this issue with Honey Encryption [10], in which the data seems plausible even when decrypting with the wrong key. But it is not applicable to every type of data.

6. Vision: Heart-Beat Puzzles

The goal is to create a scheme, which we call the heart-beat puzzle, in which the admin can “restart” the time-lock puzzle with a new key, and has a fixed time window long τ to do it (that we call a “heart-beat”), and either the data can be accessed only with the new key, or it can be decrypted with the old key recovered by any remaining available actor (e.g., the bank’s CEO)after τ (meaning that all the admins actually died).

However the above seem to be conflicting requirements: if the key is eventually available it will always be eventually available and not available only if the new key does not exist yet.

The closest known primitive is break-glass encryption [20]. Break-glass encryption is designed as an emergency decryption procedure if the user loses the key (the laptop is stolen, or the user just... dies). Using break-glass encryption, one can decrypt the previously encrypted data without any cryptographic secret possessed by the user. However, the construction relies on a hardware token which has the decryption key hardwired (and obfuscated!), and it only provides detectability, i.e. if the procedure is misused, the user will know as the illegitimate attempt is recorded in (but not stopped by) the hardware token. This could be a good solution if the one holding the hardware token cares about his own reputation. Yet, we are talking about critical banking data. Everything, including reputation, has its own price.

For heart-beat puzzles to work we need something whose “time” can be “extended”. It could be a time-lock puzzle with a trapdoor, known only to the *trusted* sysadmin who can use it to *extend* the breaking time of the puzzle.

Heart-Beat Puzzle
A heart-beat puzzle is a tuple of algorithms (Puzzle.Gen,Puzzle.Ext,Puzzle.Sol) such that:

$\{Z, T\} \leftarrow \mathsf{Puzzle.Gen}(t, s)$ takes as an input a parameter t and a solution $s \in \{0,1\}^\lambda$, where λ is the security parameter, and outputs a puzzle Z and a trapdoor T.

$\perp \leftarrow \mathsf{Puzzle.Ext}(Z, T, t)$ takes as input the puzzle Z, the trapdoor T, and an extended time t, and outputs only $\perp$.

$s = \mathsf{Puzzle.Sol}(Z)$ is a deterministic algorithms taking as an input a puzzle Z and outputs a solution s.

For heart-beat puzzles, we *first* require all the security properties of time-lock puzzles (i.e. Completeness, Efficiency, and Security). An additional critical requirement is *Unclonability*, i.e. to make the “old” puzzle unavailable to access once the puzzle has been updated (so the adversary’s progress is reset). One possible implementation, is to make use of hardware tokens, as in break-glass encryption.

Another (operational) requirement is for the sysadmin to maintain access to the puzzle at all time (so that he can update it periodically). For example, one might consider a solution based on smart contracts,[14]with the added complexity the key is only recovered when a real-world event is true where the event is that the sysadmin has not sent the heart-beat after the fixed time-shift. However, in this case, we also need some additional requirements from the blockchain [16]: How do we store the key on the blockchain? How do we model the key recovery procedure? How do we model and implement the heart-beat from the sysadmin? What if the attacker forks the chain[15] and stops the heart-beat messages reaching its destination? Would we get the lock extended in the end?

7. Conclusions

“What if they all die?”, i.e. what happens if the key people are unavailable, is an important question for bank suppliers to consider, as they must meet the European Central Bank’s regulations on the continuity of critical bank business functions.

In this paper we have discussed several candidate cryptographic solutions, i.e. escrow systems, state machine replication, secret sharing MPC, and time-lock puzzles, as attempts to address such requirements. Yet, we have argued that none of these tools is sufficient to solve our problem in their current form, and thus we may need a new tool, which we envisioned and dubbed the *heart-beat puzzle*. Intuitively, an heart-beat puzzle is a time-lock puzzle where the time-lock can be refreshed given a trapdoor. It remains an open problem whether heart-beat puzzles exist (and under what assumptions), which is an interesting challenge for the cryptographic community.

An interesting open problem would be to deal with malicious sysadmins (at least to some extent). For example, even with heart-beat puzzles, the admin could just update the password without updating the puzzle. To address this, one could consider “binding” the password updating and data re-encrypting process to the puzzle updating process. We leave this for future work.

There could be more requirements if the scenario at hand is even more complicated. For instance, a new entity called the local Judge could be involved: the key could then be secret shared between the Bank, the Supplier, and the Judge of a local court; the bank can prove to the court that the supplier is not operational with them to obtain the key. Thus, another future research direction is to look further into how companies deal with our problem at the moment.

References

[1] E. Barker, “Sp 800-57 part 1 rev. 4 recommendation for key management part 1: General,” *NIST special publication*, vol. 800, p. 57, 2016.

[2] N. Bitansky, S. Goldwasser, A. Jain, O. Paneth, V. Vaikuntanathan, and B. Waters, “Time-lock puzzles from randomized encodings,” in *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, 2016, pp. 345–356.

14. Ethereum: “A built-in fully fledged Turing-complete programming language that can be used to create “contracts” that can be used to encode arbitrary state transition functions, allowing users to create any of the systems described above, and as well as many others that we have not yet imagined, simply by writing up the logic in a few lines of code.” [7]

15. The blockchain fails to achieve consensus and the nodes maintain different views of the system.

[3] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, no. 1999, 1999, pp. 173–186.

[4] Y.-J. Chang, W. Zhang, and T. Chen, "Biometrics-based cryptographic key generation," in *2004 IEEE International Conference on Multimedia and Expo (ICME)(IEEE Cat. No. 04TH8763)*, vol. 3. IEEE, 2004, pp. 2203–2206.

[5] R. Cramer, I. Damgård, and U. Maurer, "General secure multi-party computation from any linear secret-sharing scheme," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2000, pp. 316–334.

[6] J. Daemen and V. Rijmen, "The block cipher rijndael," in *Smart Card Research and Applications, This International Conference, CARDIS '98, Louvain-la-Neuve, Belgium, September 14-16, 1998, Proceedings*, 1998, pp. 277–284.

[7] Ethereum, "A next-generation smart contract and decentralized application platform," 2015.

[8] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *Foundations of Computer Science, 1987., 28th Annual Symposium on*. IEEE, 1987, pp. 427–438.

[9] C. G. Günther, "An identity-based key-exchange protocol," in *EUROCRYPT*, 1989, pp. 29–37.

[10] A. Juels and T. Ristenpart, "Honey encryption: Security beyond the brute-force bound," in *EUROCRYPT*, 2014, pp. 293–310.

[11] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982.

[12] J. Liu, T. Jager, S. A. Kakvi, and B. Warinschi, "How to build time-lock encryption," *Des. Codes Cryptogr.*, vol. 86, no. 11, pp. 2549–2586, 2018.

[13] G. Malavolta and S. A. K. Thyagarajan, "Homomorphic time-lock puzzles and applications," in *Proc. of CRYPTO*, 2019, pp. 620–649.

[14] F. Massacci, C. N. Ngo, J. Nie, D. Venturi, and J. Williams, "Futuresmex: Secure, distributed futures market exchange," in *2018 IEEE Symposium on Security and Privacy (SP)*, vol. 00, pp. 453–471. [Online]. Available: doi.ieeecomputersociety.org/10.1109/SP.2018.00028

[15] A. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[16] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[17] R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock puzzles and timed-release crypto," Cambridge, MA, USA, Tech. Rep., 1996.

[18] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[19] A. D. Santis, Y. Desmedt, Y. Frankel, and M. Yung, "How to share a function securely," in *STOC*, 1994, pp. 522–533.

[20] A. Scafuro, "Break-glass encryption," in *IACR International Workshop on Public Key Cryptography*. Springer, 2019, pp. 34–62.

[21] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[22] A. C. Yao, "Protocols for secure computations (extended abstract)," in *Proc. of IEEE FOCS*, 1982, pp. 160–164.