Security in Blockchain Applications

Prof. Dr. Peter Roßbach, Frankfurt School of Finance & Management

The current affair about Spectre and Meltdown, which affects many of the processors in use, clearly shows that there is no real security in the IT area. Security is always gradual, and the degree of security is highly dynamic. A high level of security today can drop to a low-level tomorrow if the corresponding critical security hole is discovered. The question is not whether a vulnerability exists – it is a well-known fact that there is no perfect software - but when a vulnerability will be discovered.

But what happens when the "bad guys" discover and exploit the vulnerability first? Particularly, for systems whose security mechanisms are only implemented in the software, there is a danger that they will get into an unforeseen and uncontrolled state and cause considerable damage.

Security and cryptography specialist Bruce Schneier states:"If you think technology can solve your security problems, then you don't understand the problems and you don't understand the technology." [Schneier 2000]

However, it is one of the most important features of the blockchain concept that fundamental security mechanisms are implemented via software protocols in order to prevent manipulation and abuse of power. Especially, blockchain applications, which are constructed according to the pure Code-is-Law philosophy, are condemned to security, because in the case of a security relevant event, an effective correction mechanism does not exist. If a system is to function without human control or intervention, it must be ensured that it always fulfills its functions correctly in accordance with its requirements and that it cannot be influenced externally. However, this contradicts the above-mentioned assumption that such a level of security is not possible.

Inspired by this discussion, this article deals with the security of blockchain applications. For this purpose, a 4-layer architecture of a blockchain system (see Figure 1) is developed, and the security aspects are discussed on the basis of these layers.

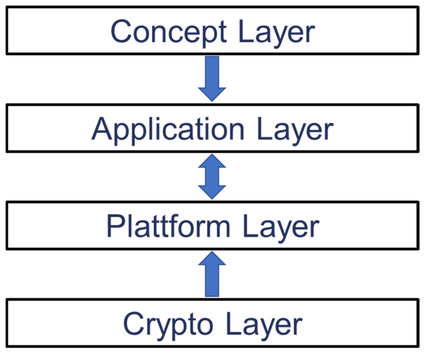


Figure 1: 4-Layer Architecture of a Blockchain System

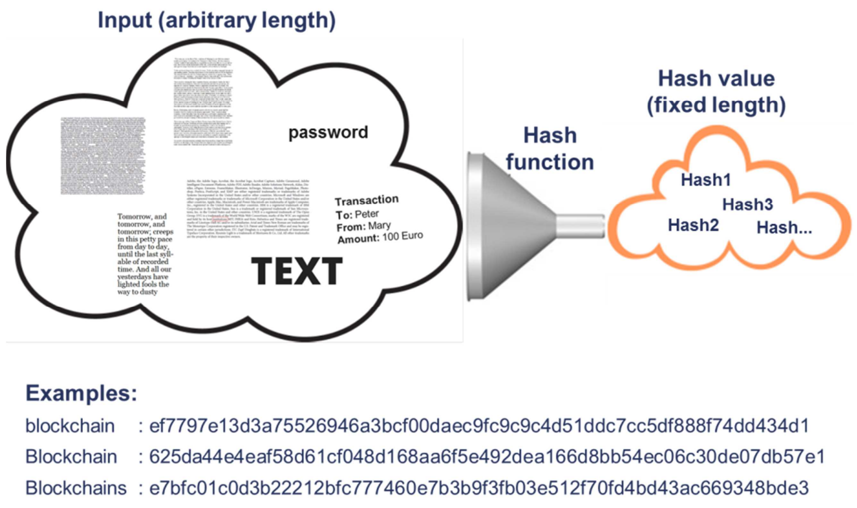
The aim is to show at which levels security problems can arise, to sharpen the necessary security awareness and to stimulate a discussion about the further development of the blockchain concept. In the following, the term security is not only considered in the narrower sense of attacks on a system, but also in the broader sense, which includes the functionality of the system. This broader concept of security therefore also includes the consequences of errors of different causes, which can lead to the system not functioning as intended.

Due to the distribution and popularity of the platform, this contribution refers primarily to Ethereum in the technical parts, which can be considered as the most important representative of the implementation of a blockchain approach in the narrower sense.

The Crypto Layer

One of the constitutive elements of blockchain architecture is cryptography. In most publications on this topic, it is assumed that the cryptographic methods used are (sufficiently) secure. Security in cryptography is based on the fact that no algorithms for solving specific mathematical questions have been found to date [Esslinger 2014], e.g. the discrete logarithm problem. In other words, security in cryptography is based on the insolvability of these mathematical questions. The consequence is that an attacker only has the chance to apply trial and error. By determining the number of possible solutions, e.g. by specifying the key length, the necessary time requirement can be set so high that finding the correct solution with the given technical possibilities is not possible in an acceptable time. Thus, the cryptographic methods are considered safe as long as (a) no algorithms are found to solve the mathematical questions without difficulty or (b) the necessary computing power is not available to find the correct solution in reasonable time.

Blockchain concepts are essentially based on two cryptographic approaches: hash functions and asymmetric cryptography.



Examples:

```
blockchain   : ef7797e13d3a75526946a3bcf00daec9fc9c4d51ddc7cc5df888f74dd434d1
Blockchain   : 625da44e4eaf58d61cf048d168aa6f5e492dea166d8bb54ec06c30de07db57e1
Blockchains  : e7bfc01c0d3b22212bfc777460e7b3b9f3fb03e512f70fd4bd43ac669348bde3
```

Figure 2: Hash Function

Hash functions map an input of arbitrary length to a sequence of characters with fixed length (see Figure 2). The same input always has the same output (hash value). Hash functions are designed in such a way that similar inputs lead to very different outputs. They are one-way functions, i.e. it is not possible to recalculate from output to input. Since an unlimited number of inputs are mapped to a limited number of outputs, this concept theoretically results in an infinite number of inputs for every possible hash value.

This opens up the possibility of so-called collision attacks, in which two different inputs having the same hash value are sought. For example, last year Google utilized the hash function SHA-1, which is still widely used today, to calculate a successful collision using two very similar-looking PDF documents. Successfully finding a collision would open up the possibility of referencing one of the documents by the hash value and later claiming that it was the other document belonging to the hash value.

Another, technically much more difficult, type of attack is the so-called preimage attack, in which an alternative input is sought to an existing combination of input and hash value. A successful search would allow generating an alternative truth. However, the resulting alternative input must be plausible in the context of the application, which leads to an increase in the degree of difficulty. For example, an alternative input to the hash value of a transaction would also have to be interpreted as a transaction data record otherwise the alternative would make no sense.

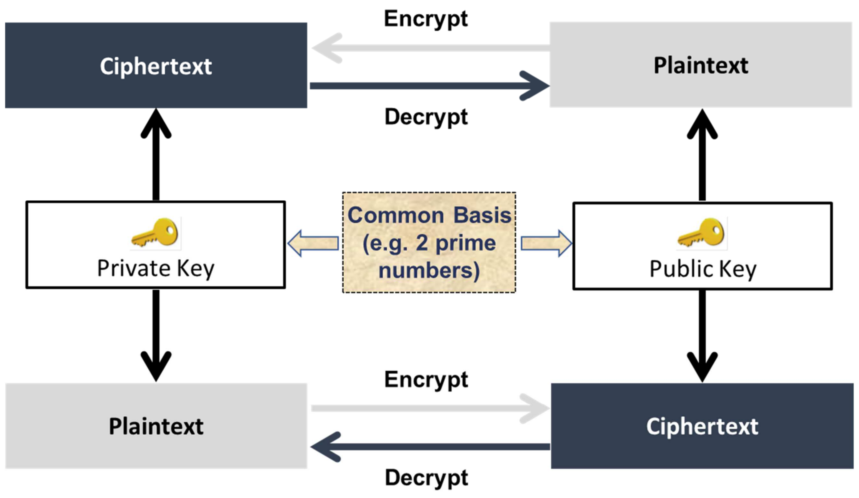


Figure 3: Principle of Asymmetric Cryptography

Asymmetric cryptography is based on the fact that two keys, a public and a (secret) private one, are used for encryption and decryption operations. The private key can be used for both encryption and decryption (see Figure 3). The public key then has the opposite role. Thus, a message encrypted with the public key can only be read by the holder of the private key, and vice versa [Shenk 2018]. The primary security feature here is that, even that they are related, the private key cannot be derived from the public key in an acceptable time since the private key is the key to the rights that one has in blockchain systems, for example, to authorize a transaction.

The possibility of discovering algorithms to break the used cryptographic methods in a very concise time is the Sword of Damocles in cryptography. However, the constant increase in computing power is also a problem here, as it accelerates the trial process. Asymmetric cryptographic methods are also endangered by the development of quantum computers. Their successful development would mean the end of currently used asymmetric methods. The hash processes are considered to be significantly more resistant to this technology [Esslinger 2014]. When this type of computer will be available is controversial among the experts. The majority of estimates range from 5 to 25 years. But even in this case, quantum-safe cryptographic methods are already being developed today.

In blockchain applications, the corresponding cryptographic methods must be modified with regard to the key length or replaced as the risk of attacks increase. In these cases, the blockchain itself must be converted to be compatible with the new method. In contrast to a centralized system, this is a much more complex undertaking for a distributed system. It would have to be organized and ensured that the distributed blockchain instances are converted to remain part of the system. Those who keep copies of the old instances can decrypt cryptographically protected legacy data that may be stored on the blockchain after the method has been broken with the consequence of having access to secret information. This is an essential challenge for the designers of blockchain applications.

But the problem is not only the security of the algorithms themselves. As the Achilles' heel of IT security in this area, the implementation of the cryptographic procedures has repeatedly emerged. The fact that the correct implementation is not trivial can be stated by the frequently discovered security holes in the common cryptographic software libraries, which are due to implementation errors. Examples are the Heartbleed bug [http://heartbleed.com] or the vulnerabilities in the TLS implementations [Ermert 2017].

In addition, deliberate backdoors can also be installed for various reasons, which at first only serve the originator, but when discovered by strangers an immense amount of damage could be the consequence. A famous example here is the backdoor in the random number generator Dual_EC_DRBG based on elliptical curves, which was even a standard of the National Institute of Standards and Technology (NIST) [Esslinger 2014 and Ermert 2017].

When selecting a blockchain platform, care must be taken to ensure that it uses well-known and quality-assured cryptographic libraries. In this area, open source libraries are preferable, which are not a guarantee of faultless content, but which can be assumed to detect errors earlier due to the transparency of the code and close them more quickly.

In conjunction with the next layer, which uses the cryptographic functions for the blockchain functionalities, the interaction between the two layers must also work correctly. This includes the error-free integration and use of the cryptographic libraries, to avoid corresponding security problems at this point.

The Platform Layer

This layer is the development and execution platform for blockchain applications. It provides the tools for the development of blockchain applications and ensures their operation. One example is Ethereum.

The platform layer is particularly important concerning security requirements. Security vulnerability at this level can be very severe since it not only affects a single blockchain application but all blockchain applications affected by the issue.

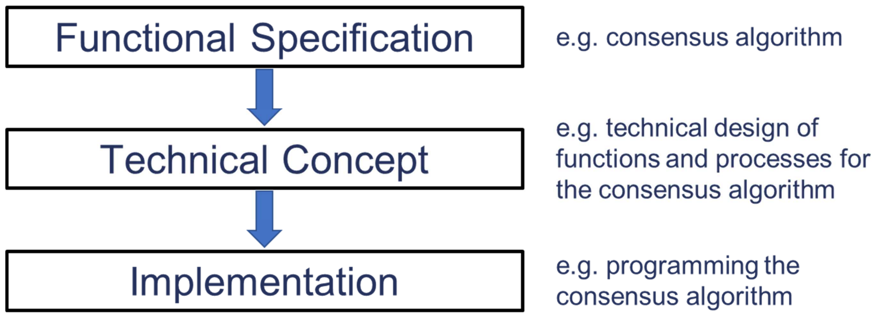

Functional Specification
e.g. consensus algorithm
Technical Concept
e.g. technical design of functions and processes for the consensus algorithm
Implementation
e.g. programming the consensus algorithm

Figure 4: From Concept to Implementation

Security problems can arise from the conception, as well as the implementation (see Figure 4). Concerning the conceptual design, it is possible to distinguish between the functional specifications and the technical concept. The functional specifications include the design of the elements of the blockchain system, e.g. the choice of the consensus mechanism and its design. The technical concept is the transformation of the functional specifications into the corresponding technical processes and functions. These must then be programmed. Errors can occur at all three levels, leading to security problems. The higher the level at which they occur, the more critical an error is, because it must be fixed in all the levels below.

As with any other software system of this size and complexity, Ethereum repeatedly detects software bugs and then closes them with the appropriate bug fixes. Depending on the type of error, more or only a few of the Smart Contracts running on the platform are affected. Since the platform is open source, the discussions about the bugs and how to fix them mostly take place in public forums, such as Reddit [http://www.reddit.com/r/ethereum]. Here, the developers of blockchain applications can inform themselves promptly and react accordingly.

A problem arises in this context when the vulnerability is discovered by someone who badly exploits it before it becomes known. An example of this is a vulnerability discovered by an unknown person in the multi-signature function of the wallet from Parity Technologies in July 2017. It existed for months and was exploited after its discovery resulting in the theft of Ethereums cryptocurrency ether. Fortunately, other participants of the network noticed the activities. They “stole” the majority of the affected ethers themselves to later return them. Although the wallet is a third-party product, it has affected many users of the Ethereum network due to its distribution. In December of the same year, a vulnerability was discovered in the same software, which resulted in a larger amount of ether to be frozen until the vulnerability was closed.

The example documents the fundamental problems regarding the provision of secure platform services in the blockchain area. But even in the hypothetical case that the platform is always secure, it still has a significant impact on the security of the applications developed with it. The platform's developers are responsible for the processes, functionalities, and tools it provides. This mainly includes the instruments for the development and operation of blockchain applications, which are then needed for the next layer, the application layer.

The Application Layer

The application layer includes all activities to develop and operate a blockchain application. The tools required for this are provided by the specific platform or by third parties.

At Ethereum, for example, Solidity is the programming language for developing smart contracts. As a programming language, Solidity is similar to the C languages, such as C and Java, which makes it easier for developers to get started. Due to the special blockchain environment, Solidity contains additional constructs that are necessary for the operation of Smart Contracts in a blockchain environment.

Compared to conventional programming languages, however, Solidity has some limitations, at least in its current state. As a consequence, even experienced programmers often find themselves in situations where they are not able to realize things as they have learned or to which they are accustomed. These include missing floating point arithmetic, limitations on return values and types, and the lack of functions for general error handling and so on. Correspondingly, a trial-error procedure often is applied, which hinders systematic and quality-oriented development. In addition, there is a lack of documentation, so one often depends on forums and blogs.

Concerning the blockchain-specific components of Solidity, some of the constructs are difficult to understand or handle. They require knowledge and experience by the programmers to be adequately mastered. Ethereum's high development speed also leads to frequent changes that can affect the development processes.

Due to this complexity, the risk of making mistakes in the development of blockchain applications is high. Even experienced developers make serious mistakes, as has already been pointed out in the case of the multi-signature function. Another example is the incident around TheDAO, where a Smart Contract did not work as intended. In a study conducted in February 2018, researchers found thousands of weak points in Smart Contracts running on the public Ethereum blockchain using relatively simple search methods [Nikolic 2018].

An additional problem with Ethereum is that a faulty Smart Contract simply cannot be replaced by a corrected version. According to the philosophy, the code is irreversible. To correct an error, the faulty Smart Contract must be technically "destroyed" and changed to another error-cleaned Smart Contract. However, this must be taken into account in the basic architecture of the application. For this reason, it may be necessary to introduce additional Smart Contracts, which allow managing the system, such as switching to a new Smart Contract. This means that the architecture of a blockchain application may become more complex for these reasons.

Increasing complexity also broadens the risk of making mistakes. Consequently, blockchain applications require meticulous development processes with explicit security-conscious programming and extensive quality tests to avoid errors that could lead to security problems.

Nevertheless, it must be assumed that errors and security problems may still exist. Again, the fundamental problem is that there is no guarantee that those who are responsible for the blockchain application will be those who discover the security problem. If they are informed in time, they can react. However, if the vulnerability is discovered by strangers who exploit it before it can be closed, it becomes much more complicated.

Strict adherence to the principle of irreversibility in blockchain applications would inevitably be accompanied by a code-is-law philosophy, which would mean that damage caused by a security incident would have to be accepted. Otherwise, correction processes would have to take place that are carried out by the appropriate responsible authorities. Such a possibility of intervention is connected with power.

If it lies in the hands of a central instance, there is a conflict with a basic property of the classic blockchain concept. If this power is to be distributed, a concept must be found in which the decision-making and reaction process is not too lengthy and in which the outcome is certain. This is not the case with the current community decisions.

The Concept Layer

The possibility for errors in blockchain applications does not only start with the implementation but could already have its roots in the conception. Insufficiently deliberated concepts or errors in them have even more severe consequences, as they cannot be simply corrected by a software update.

The causes of conceptual errors or deficiencies may be manifold. For example, the designers of a blockchain application may not have a sufficient understanding of the underlying technology. One example would be an identity management system where the identity in the blockchain is referenced solely by the hash value of the scan of an ID document. With sufficient computing capacity, it is possible to generate a scan of another ID document with the same hash value (see also the explanations in the section "Crypto Layer"). The reason for this is that a scan is stored as an image file that has a metadata area. This area may be modified in many ways without any effect to the visible image. It provides a perfect playground to search for an input with the same hash value. The procedure is similar to bitcoin mining.

Another cause may be inadequately deliberated consequences of the application concept. An example of this is a passenger transport service whose prices are functionally dependent on the demand. In the event of a catastrophe, such as an earthquake, this may cause prices to explode and consequently only transport those who have enough money [Buhl 2017]. In this case, there is also an additional ethical dimension.

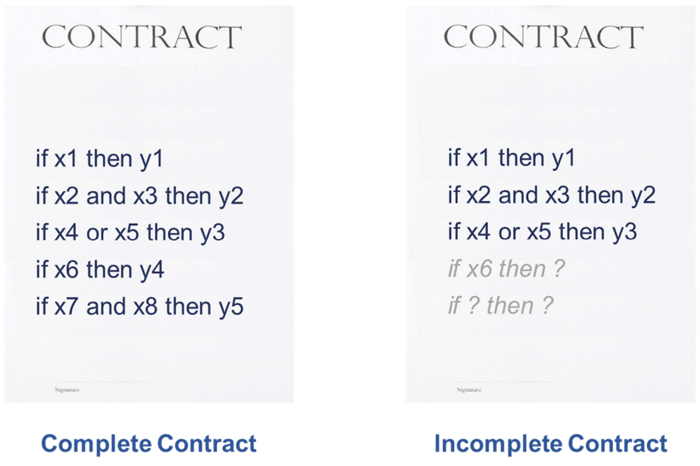


Figure 5: Complete and Incomplete Contract

The economic contract theory also distinguishes between complete and incomplete contract types [Wikipedia 2017]. An incomplete contract is a contract between partners in which not all contingencies can be contractually determined or taken into account in advance (see Figure 5). In the case of incomplete contracts, it is therefore not possible to include precautions for all possible future conditions ex-ante, since in the course of time requirements or events may occur that were not yet known at the time of contract creation. The reasons for this may be the complexity of the subject matter and/or the unpredictability of future events.

In accordance with this distinction, Smart Contracts are only suitable for applications in which complete contracts are existent [Davidson 2016]. In all other cases, the concept would have to be designed so that contract-relevant facts may also be regulated outside the Smart Contract. On the one hand, this is in contradiction to the blockchain philosophy. On the other hand, it implies decision-making authorities outside of the technical implementation since in such cases human evaluations are required to be able to handle unforeseen issues. Furthermore, it is not always certain in advance whether a contract is a complete contract or not. As a result, blockchain constructs, at least in the current state of development, are particularly suitable for simple applications where both the underlying contracts and the resulting Smart Contracts are not very complex.

Conclusion

The explanations have shown that there are various sources of possible security problems across the four layers of a blockchain application defined here. Due to the specific design of blockchain applications, these are systems with a high degree of autonomy. Security is therefore essential. Accordingly, the security requirements must be met with quality and safety oriented design and development processes. Particular attention must also be paid to the quality of the developers, as they are in a different environment when developing blockchain applications as opposed to conventional applications. The current maturity of the development instruments and execution platforms represents an additional hurdle.

Given the fact that security is always gradual and not static, the question arises as to whether the purist blockchain concept, in which the security mechanisms are almost completely implemented at the software level and no effective correction mechanisms exist, is suitable for the various application potentials that are currently being discussed. This applies in particular to applications that are of systemic importance, such as in energy or finance. The failure of such a system could lead to unpredictable states. In this respect, there is also an urgent need to further develop the concepts into those with governance structures that can if necessary, quickly and effectively react to combine the advantages of the blockchain concept with the security requirements in specific areas of application.

Particularly in the case of complex and ex-ante not completely determinable areas of application, there will also be demands on the adaptability of the system. The development in the area of autonomous systems shows that due to increasing complexity and uncertainty at development time the opportunities for adapting and changing the systems are more and more shifted to runtime [Schneider 2017].

Literature:

Buhl, H. U. (2017): Blockchain-Technologie als Schlüssel für die Zukunft?, http://www.youtube.com/watch?v=Z0niGKbPBHA

Davidson, S.; De Filippi, P.; Potts, J. (2016): Economics of blockchain, http://ssrn.com/abstract=2744751.

Ermert, M. (2017): Quantenkrypto nur eine Ablenkung?, in: c't, Heft 17, S. 34-36.

Esslinger, B. et al. (2014): Krypto + NSA = ?, in: <kes> Die Zeitschrift für Informationssicherheit, Heft 1, S. 70-76.

Nikolic, I. et al. (2018). Finding the Greedy, Prodigal, and Suicidal Contracts at Scale. http://arxiv.org/pdf/1802.06038.pdf.

Shenk, M. (2018): The Quantum Countdown - Quantum Computing And The Future Of Smart Ledger Encryption, Z/Yen Group, http://www.longfinance.net/DF/Quantum_Countdown.pdf.

Schneider, D. et al. (2017): Umfassende Sicherheit – Safety und Security im Kontext autonomer Systeme, in: Informatik Spektrum, Band 40, Heft 5, S. 419-429.

Schneier, B. (2000): Secrets & Lies, John Wiley & Sons.

Wikipedia (2017): Contract Theory, http://en.wikipedia.org/wiki/Contract_theory.