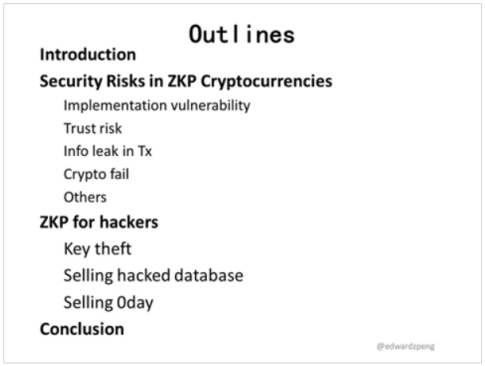
Security Risks in Zero Knowledge Proof Cryptocurrencies

12月20, 2019

Zhiniang Peng of Qihoo 360 Core Security

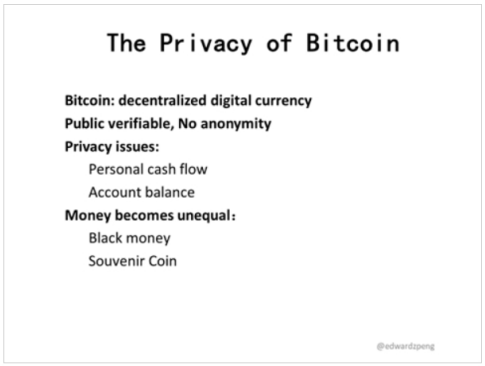
Here I am publishing my talk in PacSec2019 with slides and Speech. It's a survey of security risks in zero-knowledge proof cryptocurrency. I also propose some ZKP application for hackers to evade supervision. Hope You find it useful.



Here is the outlines of my talk. First, I will briefly introduction zero knowledge proof in cryptocurrency.

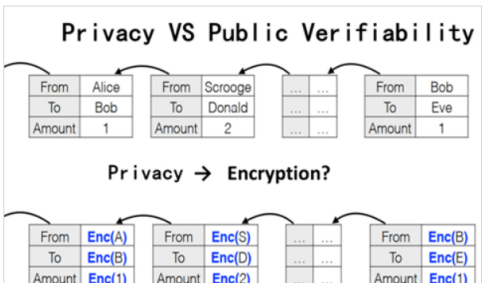
Then I will talk about some security risk in ZKP cryptocurrency. Including: implementation vulnerability, trust risk, information leakage in transaction, crypto failure, and some others security issues.

After that, I will give some application of ZKP for hackers. For example, how to sell a secret key, a hacked database, or sell some 0day using zero knowledge proof to prevent you self to be exposed.



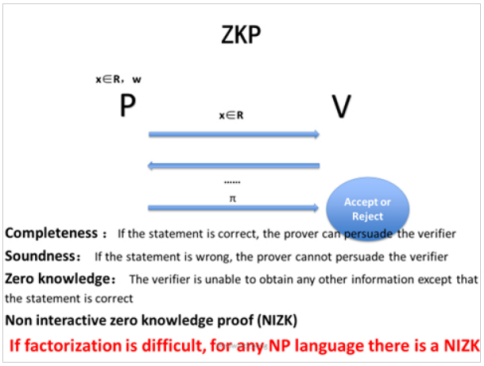
Everyone knows about bitcoin. Bitcoin is a decentralized digital currency. All the transaction of bitcoin is published on the internet, so it can be public verifiable. However, there is no anonymity for Bitcoin. All your personal cash flow, your balance of your wallet address is public online, everyone can read it. So, here is no privacy in bitcoin. Bye the way, because all the history of bitcoin is public. So, each money become unequal. For example, some money maybe used by some the shadowbrokers before, so the money become black money. Or some money maybe used by Satoshi Nakamoto, this money become a souvenir coin.

The privacy issues and the unequal issues is bad for a currency system, researcher is working hard to solve these issues.



Here is a typical cash flow of bitcoin. As we can see in the picture above, Alice send 1 bitcoin to bob, and finally bob send this 1 bitcoin to Eve. All the detail is published on the chain. How can be solve the privacy issues? A simple idea is just encryption all the transactions.

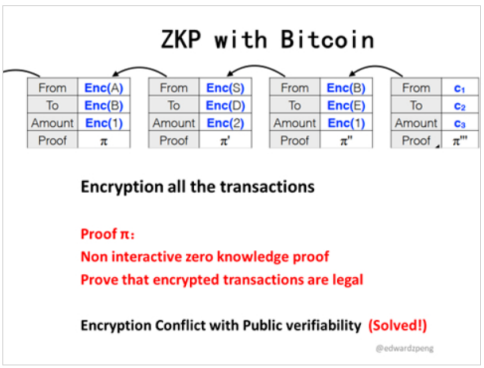
For example, we can see the following picture. We can encrypt the sender address, the receiver address, and the amount of money. Then your privacy is protected. But this cannot work, because if you encrypt everything. Other people cannot verify the transaction, this will break the Public verifiability of bitcoin.



To solve this problem, here come zero knowledge proof (ZKP). In a ZKP system, there is a prover P, and a verifier V. P have some statement x, and he claim to the V that x belong to some language R. After some interaction, P will send a proof pi to the V. Then V can decide to accept the statement or reject it.

There are 3 property of a ZKP system. First, Completeness： If the statement is correct, the prover can persuade the verifier. Second, Soundness: If the statement is wrong, the prover cannot persuade the verifier. Third, Zero knowledge： The verifier is unable to obtain any other information except that the statement is correct.

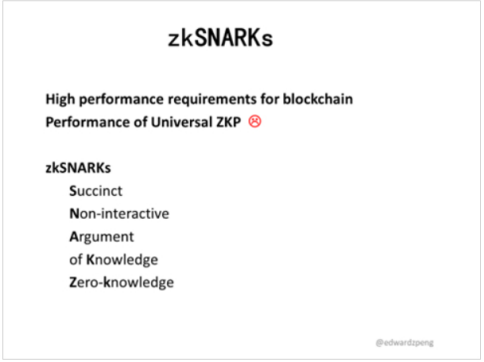
The protocol in the picture is an interactive zero knowledge proof, in a real application. P and V may not be able to online at the same time. So, we need the zero-knowledge proof to be non-interactive. Which means P can directly send a proof to V, then V can decide to accept or reject it. This is a non-interactive zero knowledge proof, we call it NIZK. It's not hard to construct an NIZK system. If factorization is hard, then for an NP language, we can build a NIZK.



After we have a ZKP system, we can add it to bitcoin to get a system with privacy.

We can encryption all the transactions, then every information will be protected. In the meantime, we also provide a proof pi. Which is a non-interactive zero knowledge proof. To prove that the encrypted transaction is legal. They are not double spending, and it follow the rule of the system.

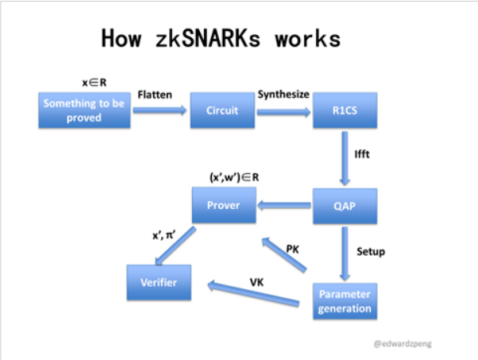
Then we solved the conflict between encryption and public verifiability. We get a bitcoin system with privacy.



Blockchain system always have a high-performance requirement. But the performance of universal ZKP system is very bad. Here comes zkSNARKs system.

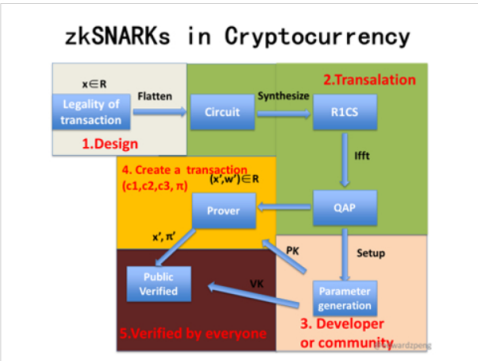
In short, zkSNARKs system is a non-interactive zero-knowledge proof system that meet the performance requirement for blockchain system. It's

widely used by some of the cryptocurrency system with privacy.



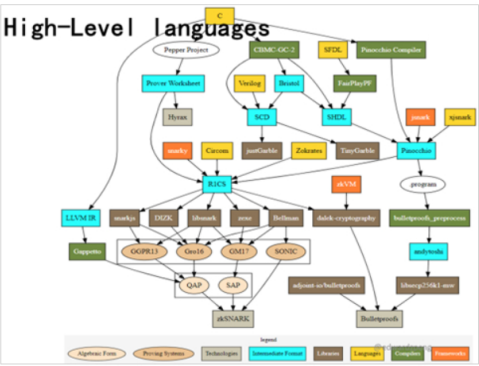
Here is a picture showing how zkSNARKs works. First, you have something to be prove. Then you can write it down in circuit language, then you translate the circuit language into R1CS system, and QAP system.

After you have a QAP system for the things you want to prove, you can setup a zkSNARKs proving system for it. After parameter generation, you have a proving key PK and a verification key VK. The prover wants to prove a statement x prime; he can generate a proof pi using the proving key PK. Then he sends the proof pi to the verifier. The verifier uses the verification the verify the proof, then he the decide to accept or reject the statement.



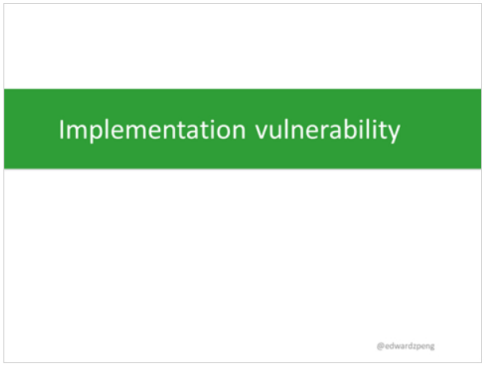
In a cryptocurrency system, zkSNARKs works like this. First, the project designer designs the protocol. In the protocol, we have some rule of a legal transaction. To prove a transaction is legal, you need to prove that the transaction follows these rules. Then you can write those rules in circuit language, and translate it into a QAP system. Then the developer or the community will do a setup to generate the parameter for the proving system. After that, we have the proving key PK and the verification key VK in the source code of the cryptocurrency system.

When someone wants to send some money to others, he can create an encrypted transaction. Then use the proving key to generate a proof pi to tell others that this encrypted transaction is legal. Then he can publish the encrypted transaction and the proof to the blockchain. Then everyone can verify and accept this transaction.

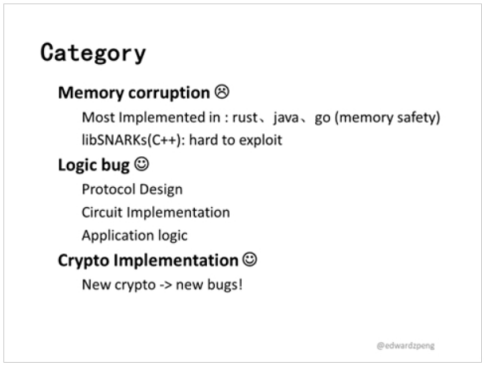


This is a picture of High-Level languages for zkSNARKs. You can see from the picture that currently there are lots of project about zksnark, most of them are using in blockchain. There will be some security problem.

Here I will talk about the current security risk in ZKP cryptocurrency.



First, I will talk about some implementation vulnerability.

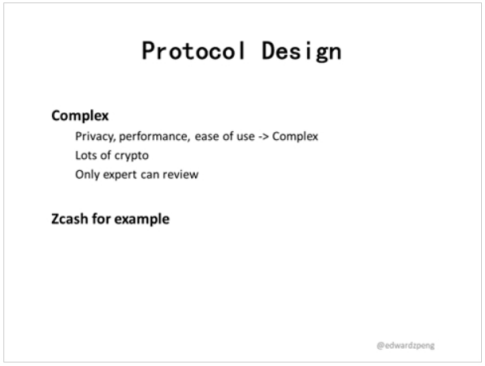


Here is a category of the implementation vulnerability in ZKP cryptocurrency.

The first kind of bug is memory corruption. However, most of the newly ZKP cryptocurrency system is implemented in language with memory safety, such as rust, java, go. So, it seems impossible to found memory corruption bug on them. There also some protects are written in C++, such as libSNARKs. However, the scenario of ZKP in those application is hard to exploit. So, memory corruption bug not works here.

The second kind of buy here is logic bug. There are many Logic bug in protocol design, circuit implementation or application logic.

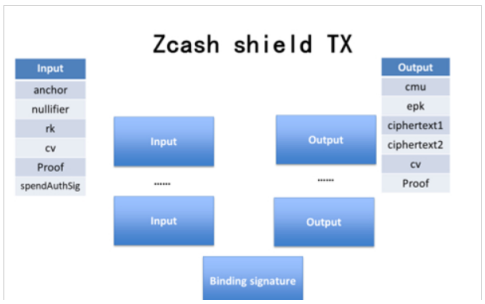
Also, there are many bugs in the implementation of crypto scheme. Because the ZKP cryptocurrency system always use some newly designed crypto, with the new crypto, there comes new bugs.



First, let's look at the protocol design.

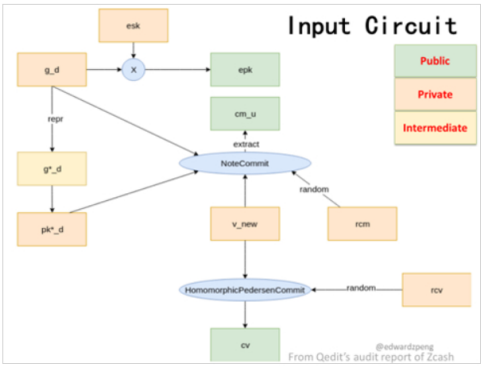
The goal of design a zkp cryptocurrency is to achieve both privacy, performance and ease of use. However, those property are conflict with each other, so the protocol will always become very complex. And involve in a lot of crypto. It's difficult to understand those protocol, and only expert can audit them.

Here I will take Zcash for example.

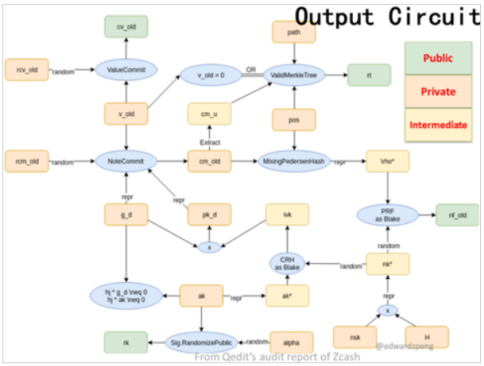


This is the structure of a shield transaction for Zcash. I am not going to detail here; I will only briefly introduce it.

In a shield transaction, there are many inputs and many outputs. You can think those input and output are encrypted. Inside each input and output, there is a proof to prove its Legality.



The input must satisfy some rule, the rule is implemented in circuit. As you can see in this picture, this is the input circuit of a Zcash transaction. Every input of a transaction should satisfy this circuit.



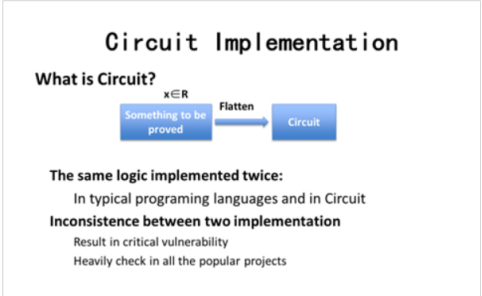
This picture shows the out circuit of a Zcash transaction. Every input of a transaction should satisfy this circuit.

This circuit is very complicate, I am not going to the detail here. Because it's almost impossible to make you understand these in a short time. If you are interested in the design, you can read the specification of Zcash. Which is page paper close to 200 pages.

Here is an example of protocol design bug in ZKP cryptocurrency. Zcash faerie gold attack. In this attack, if the attacker sends two notes with the same rho, then only one note can be used by the receiver. It can used to trick the receiver to accept a fake money. The fix of this bug is to force the parameter to be different by using some crypto technique. The detail of this bug can be found in this url.

This is just an example of the protocol design bug, you need to really look in to the design logic of the project and fully understand it.

By the way, a very similar bug also occurs in Monero.

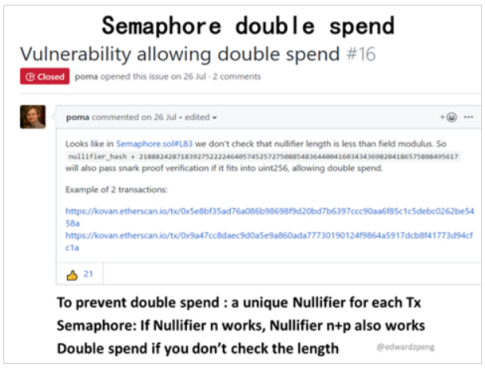


Circuit implementation is also a attack surface of ZKP project. First, you may ask what is a circuit. In a ZKP system, you have something to be proved. That is to say, you have a statement X satisfy some function F. Normally, to check the statement, you should implement the function F, and feed X into it. In a ZKP system, you also need to implement the function F in circuit language, so it can be converting to a proving system. Implement something in circuit is just like make a hardware circuit, it's very easy to get things wrong. I So you need to implement some logic twice, both in a typical programing language and in circuit.

If there is an inconsistence between the two implementations, there will be a critical vulnerability. However, all the popular projects have done a lot test to prevent this kind of bug from happening. So, I did not find Low-hanging fruit bug. But this is a big attack surface for ZKP application. If you want to found some bug in ZKP project, you should spend some time on this.

To build a ZKP application, application developer can simply use some ZKP library. However, due to the lack of sufficient understanding of ZKP technology, their code is easy to be vulnerable.

Here I will give some example of bugs in ZKP application.

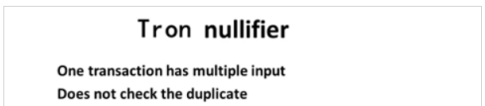


The first example is the Semaphore double spend attack.

In a typical ZKP cryptocurrency system, to prevent double spend, you need to bind a unique nullifier for each new output. Then publish the nullifier when you want to spend that output. However, in semaphore, the bind is broken. Because it doesn't check the length of the nullifier. If nullifier n works, the nullifier n plus p also works. Here p is the modulus of the group. So, attacker can double spend one output many times with different nullifier.

Here is a similar bug I found in Tron. Tron immigrate the Zcash shield transaction to their system. In their implementation. They choose to use librustzcash instead of writing their own ZKP library.

However, when using librustzcash. Tron directly use the functions in the lib without add constrains on the variables, just like the semaphore example, attacker can easily cheat the lib to achieve double spend attack.



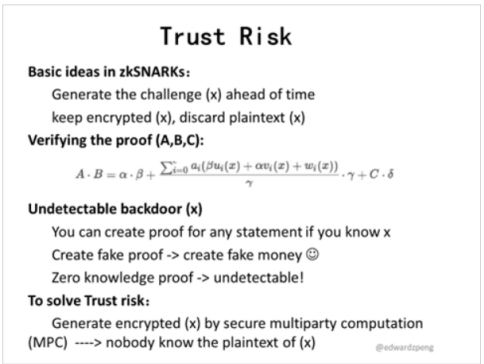
Here is another bug we found on Tron. As we show before, there are multiple input and output in a transaction. To prevent double, spend, you should check every input is different. However, Tron did not check the duplicate of the nullifier of a transaction. Attacker a leverage bug to double spend any output he receives.

Beside logic bugs, there are also many bugs in crypto implementation of an ZKP project. There ZKP project always use a lot of new crypto scheme, and implement these new crypto schemes will always result in new bugs.

For example, there is a soundness error vulnerability in libSNARKs. In the early version of LibSNARKs, the R1CS to QAP reduction had a bug. If the QAP polynomial s not linear independence, the reduction will result in soundness error. Which means attacker may create a fake proof for any statement. Detail of this bug can be found on this paper.

Another example is the Zcash side channel attack. Which is found by researcher from Stanford university. When a Zcash node processing a transaction related with himself. He will try to decrypt the transaction. However, there is a side channel bug in the decryption process. If an attacker relays a malicious transaction related to address A to a node, if address A is belong to this node, then the decryption process will be very slow. This will reveal that the node is related to the address, and break the anonymity of the system. Details of this attack can be found on this paper.

Here come second security risk of ZKP cryptocurrency. The Trust risks.

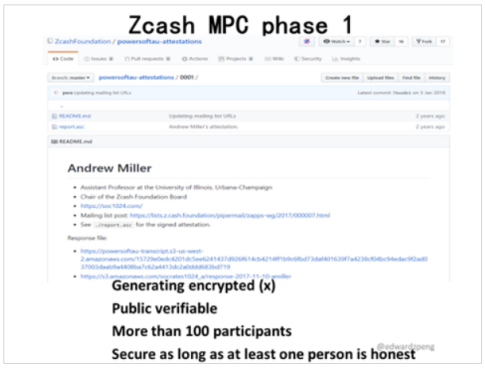


The basic idea of a zkSNARKs system is that: to generate the verifier's challenge x ahead of time. Then we keep an encrypted form of x, and discard the plaintext x. Then everyone can reuse the challenge to the verify the proof.

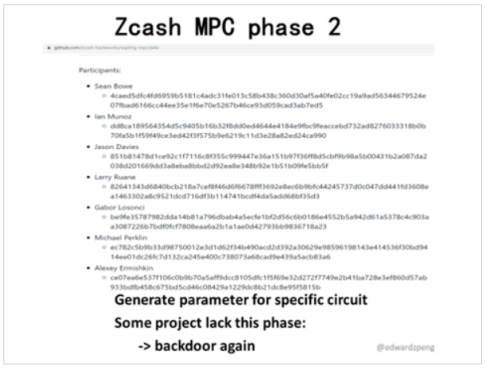
When verifying the a proof (A,B,C), we are actually check the following equation in encrypted form. Without known the plaintext of the parameter, only the one knows a witness can prove a statement.

However, if you know the plaintext of x. You can create proof for any statement. Then you can create fake proof that you have 1 million Zcash. This make you can create arbitrary fake money. And because the fake proof is also a zero-knowledge proof, so nobody can tell whether you tell a lie. It's undetectable.

To solve this trust issue, we can use secure multiparty computation (MPC) to generate the encrypted x. Then nobody knows the plaintext of x, no one have the backdoor.



For example, Zcash generate their parameter by MPC. There are two phases in Zcash MPC. Phase 1 is used to generate the encrypted x. There are more than 100 people join the MPC. All the script and log are no GitHub now, you can verify the correctness of the MPC by yourself. As long as there at least one person is honest in the MPC, we can assure the parameter is secure.



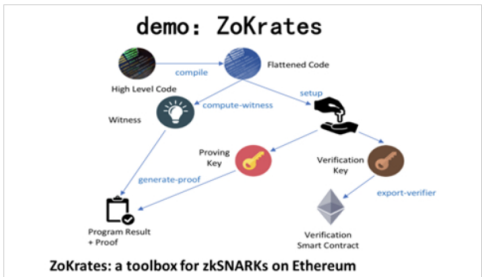
The second phase of Zcash MPC is to generate the parameter for the specific circuit. This procedure is also important. If someone know the plaintext of the parameter, it can also create a fake proof.

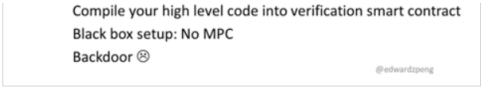
If a project lacks this phase, there will be a backdoor again.

Here I can share my personal experience in MPC. There are a lot of project are doing MPC now. For example, Ethereum, Tron. However, my personal experience in MPC is not so good.

For Zcash MPC, I email them says that I want to join the MPC. But there is no reply. As for Tron, there is no news for 3 months. For Ethereum, I have published and sign my experience online. You can check it in This URL.

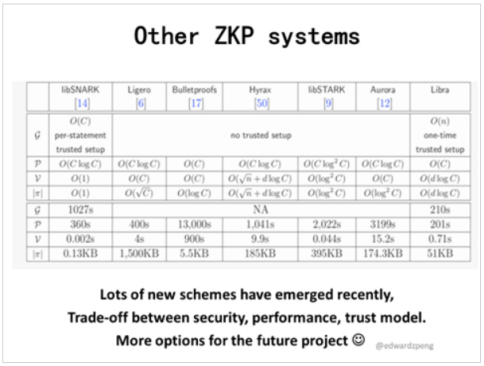
In short, MPC process is completely controlled by the organizer in a Blackbox style. Who will join is completely decide by the organizer? If you are not involved in the MPC, you cannot be 100 percent sure about its security.





There is an example of the trust risk problem. The ZoKrates. ZoKrates is a toolbox for zkSNARKs on Ethereum, you can use it to Compile your high-level code into verification smart contract.

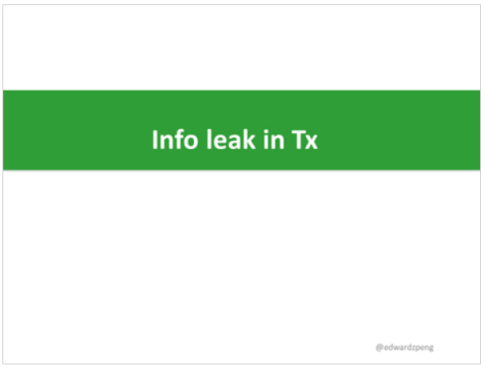
Then we can verify some proof using Ethereum smart contract. However, the parameter generation for Zokrates is black box style, there is no MPC. So, the programmer may have a backdoor for the contract.



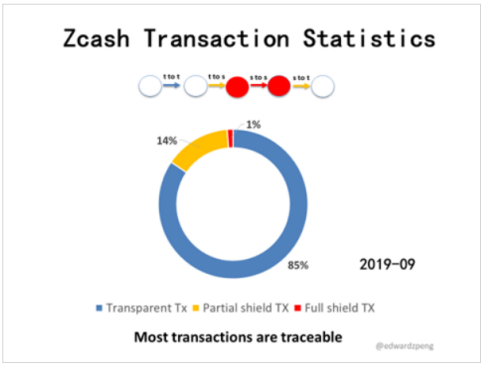
	libSNARK [14]	Ligero [6]	Bulletproofs [17]	Hyrax [50]	libSTARK [9]	Aurora [12]	Libra
$\mathcal{G}$	$O(C)$ per-statement trusted setup	no trusted setup					$O(n)$ one-time trusted setup
$\mathcal{P}$	$O(C \log C)$	$O(C \log C)$	$O(C)$	$O(C \log C)$	$O(C \log^2 C)$	$O(C \log C)$	$O(C)$
$\mathcal{V}$	$O(1)$	$O(C)$	$O(C)$	$O(\sqrt{n} + d \log C)$	$O(\log^2 C)$	$O(C)$	$O(d \log C)$
$\lvert\pi\rvert$	$O(1)$	$O(\sqrt{C})$	$O(\log C)$	$O(\sqrt{n} + d \log C)$	$O(\log^2 C)$	$O(\log^2 C)$	$O(d \log C)$
$\mathcal{G}$	1027s	NA					210s
$\mathcal{P}$	360s	400s	13,000s	1,041s	2,022s	3199s	201s
$\mathcal{V}$	0.002s	4s	900s	9.9s	0.044s	15.2s	0.71s
$\lvert\pi\rvert$	0.13KB	1,500KB	5.5KB	185KB	395KB	174.3KB	51KB

Although libSNARKs have the trusted setup issues, but there are many new zkp system emerged recently. This picture shows different ZKP systems, they are trade-off between security, performance, and trusted model.

So, there will be more options for future project with better security and trusted model.

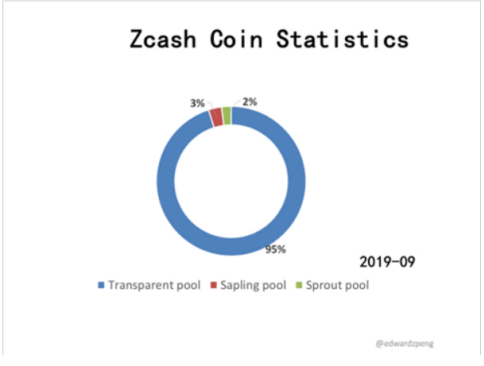


Now I will talk about the information leakage in Transactions.



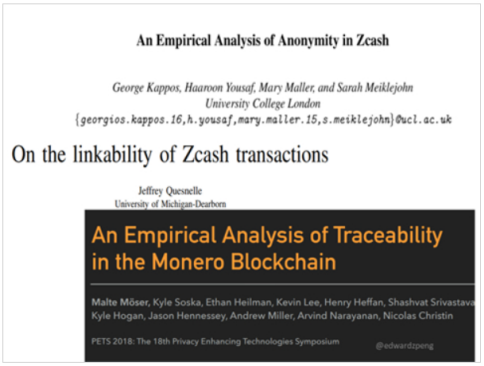
Again, take Zcash as example. We download all the transaction of Zcash in September this year. Eighty-five percent of the transaction are transparent transaction, which does not encrypt at all.

Only 1 percent of Zcash transaction is full shield transaction, fully encrypted. So, we can conclude that most of the transaction in Zcash are traceable.

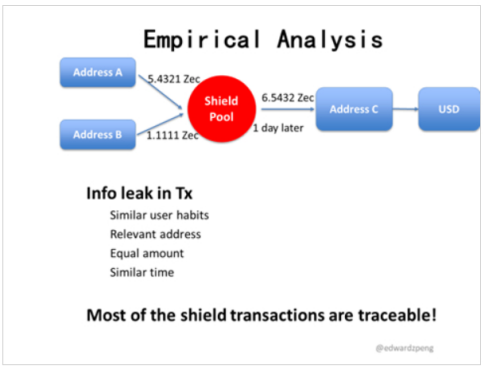


Here is the distribution of all Zcash coin. Nighty-five percent of Zcash is in transparent pool, only 5 percent of Zcash is in shield proof.

So, there will be lots of information leakage in Zcash transaction.

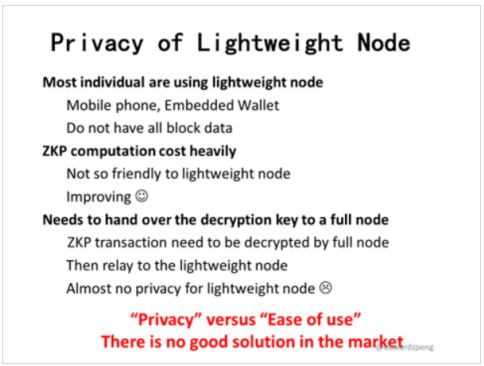


As a result, there are many papers leverage the information leakage in transparent transaction to analysis the traceability of Zcash.



Here is an example of the analysis. Suppose there is an address A, B and C. Address A and address B send some money to the shield pool in the same time. And one day later, there is a transaction send some money from the shield pool to Address C. And the amount equal to the sum of the previous two transactions. Clearly, we can infer that Address A and B are actually sending money to Address C.

There are many other information leakages in transactions. For example, the similar user habits, the relevant address, the equal amount and similar time. All of them can be used to trace Zcash. As a result, most of the shield transactions are traceable.

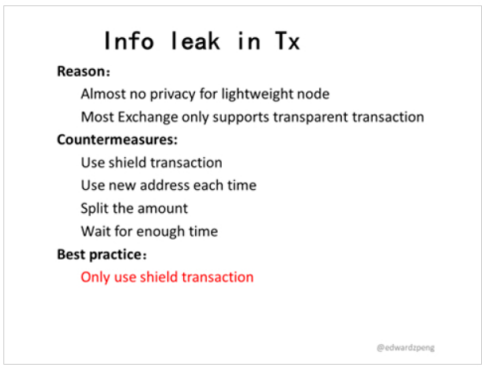


The reason for this is because most individual are using lightweight node. For example, mobile phone, embedded wallet. And the lightweight node does not have all the block data. If a lightweight node wants to decrypt transaction, he needs to hand over his decryption key to a full node. And ZKP computation always cost heavily, so it's not so friendly to the lightweight node. So, we can conclude that there is almost no privacy for lightweight node.

This is because privacy actually conflict with ease of use. And there is no good solution in the market.

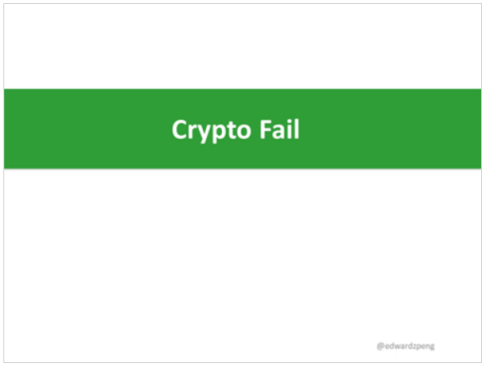
Here is an example. In Tron, if lightweight node or wallet want to receive transaction for himself. He needs to call this RPC in a full node. scanNoteIvk, in this RPC, you need to handover your decryption key in plaintext to the RPC server.

This is very unsecure. And they will not fix that, because they told me that Zcash also use the same mechanism.



In conclusion. There are many information leakages in Zcash transation because there Almost no privacy for lightweight node. And most exchange only support transparent transaction. So most of people use Zcash will use transparent transaction or partial shield transaction. Then we can use some trick to trace the transaction.

To protect your privacy, you can use shield transaction, use new address every time, and split the amount, and wait for enough time for each transaction. However, all of those cannot guarantee your privacy. The best practice to protect your privacy is to only use shield transaction. But it's hard to do it for now.

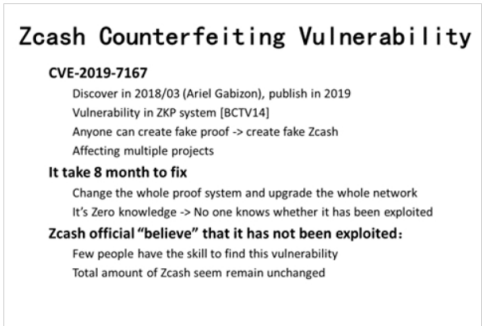


Crypto failure is also very critical in ZKP cryptocurrency.

ZKP technology is relatively new. Paper published in this year, maybe use in large scale in next year. This is very radical in my view.

And parameter selection and optimization are also very radical in some implementation. Does those parameters really secure? I think only time will tell the truth.

Although the new ZKP systems have provable security, but some of them rely on too many hard problems, and some of the hard problem are not standard one in crypto. And some of the proof itself is not well reviewed. We need more expert to audit those new system.



Here is an example of the crypto failure of ZKP cryptocurrency. Zcash counterfeiting vulnerability.

This is a bug found in March last year. But publish in this year. It's a vulnerability in ZKP system. The BCTV fourteen ZKP system. By exploit this vulnerability, anyone can create a fake proof in the ZKP system. This means that anyone can create fake money in Zcash. And this vulnerability also applies to many project forks from Zcash.

It takes about eight months to fix this vulnerability. Because Zcash need to change the whole proof system and upgrade the entire network.

There is no one knows whether this vulnerability has been exploited, because it's zero knowledge. The Zcash official "believe" that it has not been exploited. Because the "Think" there are very few people have the skill to find this vulnerability. And the total amount of Zcash seems remain unchanged.

BCTV fourteen ZKP system does not have provable security. And there is another bug found by Microsoft research before. Unlikely, it happens to be unexploitable.

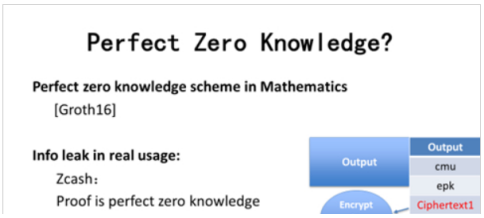
Now So the new Zcash upgrade their ZKP system to Groth sixteen, which have a provable security.

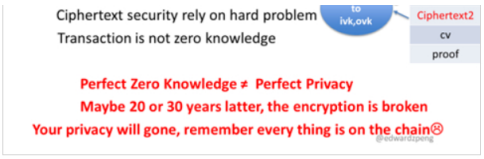
But this kind of bug is really a disaster to cryptocurrency. Will it be the last time this kind of bug happen?

Another example of crypto failure in Zcash is the hash collusion attack.

In the initial version of Zcash. The zerocash, the commitment only has one hundred twenty-eight bits, so it not computationally binds. It only has 64bits security. Attacker can use the vulnerability to double spend his Zcash, or potentially creating arbitrary amount of cryptocurrency.

Here I will also talk about some other security issues.

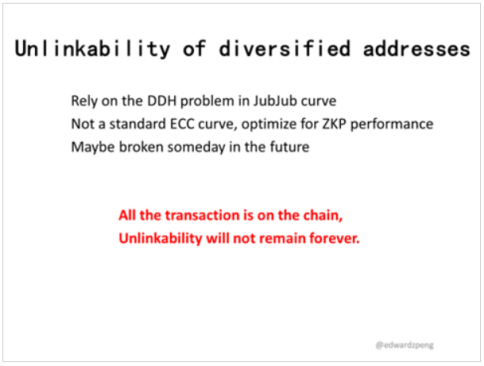




Many ZKP system is perfect zero knowledge in mathematics, However, perfect zero knowledge doesn't mean perfect privacy. There are many information leakages in real usage of ZKP.

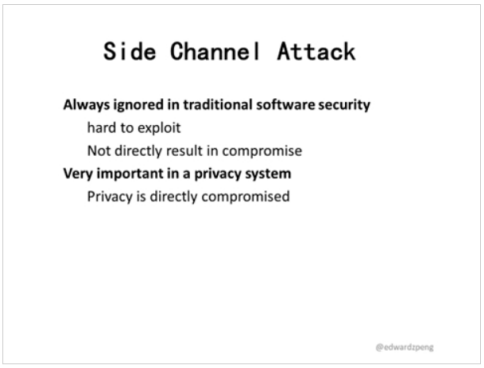
In a Zcash transaction. The proof is perfect zero knowledge. But here are many other fields in the transaction may leak information. For example, ciphertext in the transaction rely on the security of the encryption.

Maybe twenty or thirty years later, the encryption scheme will be broken. Then your privacy will be gone, because everything is on the chain.



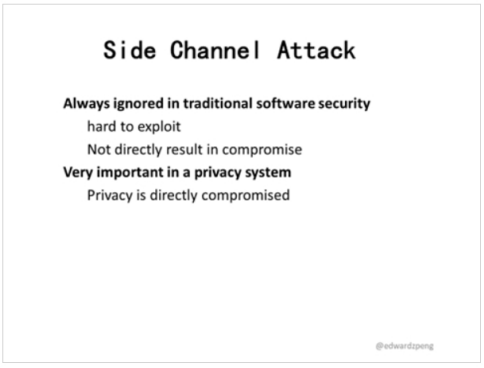
Another example is the unlikability of diversified address in Zcash. The unlikability of diversified address rely on the DDH problem in jubjub curve. This curve is a nonstandard ECC curve, which is optimized for ZKP performance. It may be broken someday in the future. Or maybe broken by a quantum computer. Because all the transaction is on the chain, the unlikability will not remain forever.

If you really care about your privacy, take care of this!



Another important thing I want to mention is side channel attack. Side channel attack always ignored in traditional software security. Because it's hard to exploit and not directly result in compromise of system.

However, it's very important in a privacy system. Because in privacy is directly compromised by a side channel attack.



Here I found out that the groth sixteen ZKP system is extremely vulnerable to side channel attack.

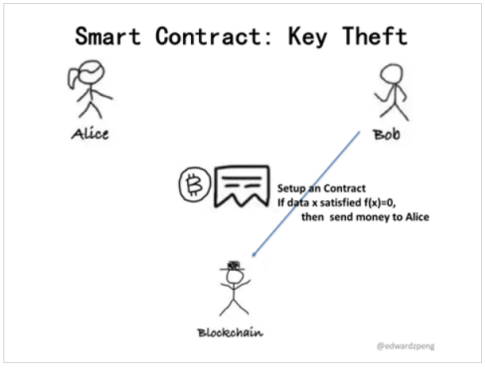
When a prover wants to proof something use Groth sixteen. He needs to calculate A, B, and C with this equation. In this equation, the computing time of A, B, C strongly related with private input ai. Which is the secret the prover wants to hide. So, an attacker can easily recover some information of the secret by launching a timing side channel attack.

There is also other side channel attack can apply to groth sixteen, for example, the cache side channel attack.

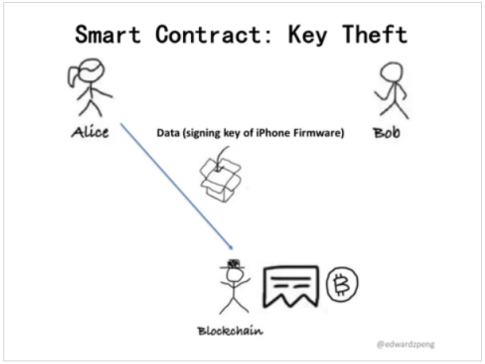
Ok, that's all for the security risks of ZKP cryptocurrency. Here we come to the second part of my talk. ZKP for hackers. Here I will give some application of ZKP can be used by hackers.

Suppose Bob is bad guy, he wants to buy the signing key of authority. Maybe he wants to buy the signing key of I phone's firmware. So, he found Alice on the dark net who claiming that she has the key.

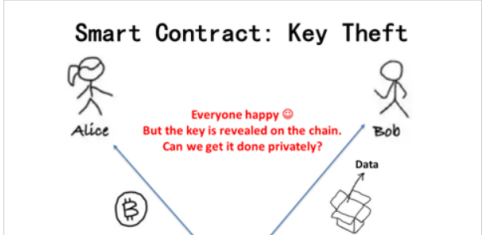
However, Alice and bob doesn't trust each other, and there is no trusted third party can help them to finish the deal. So, how can they fairly make a deal without trust?



Here is a solution. Bob can setup a contract to Ethereum. In the contract, it will verify if Alice's input x satisfied f(x)=0, which means x is the real signing key of Iphone's firmware. If it is, then send the money to Alice.

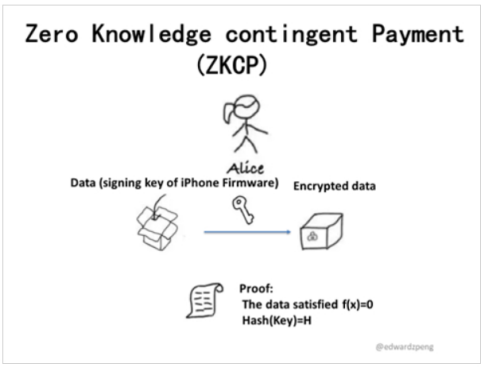


After the contract deploy in to blockchain, Alice then can send the signing key of Iphone's firmware to the blockchain.

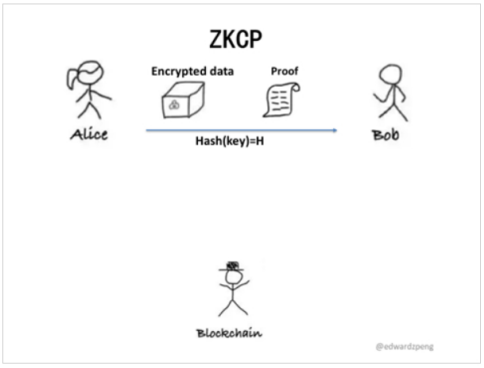


Then the contract verify the key a real one, and send the money to Alice.

Then everyone seems happy. But the key is revealed on the blockchain. Can we get it done privately?



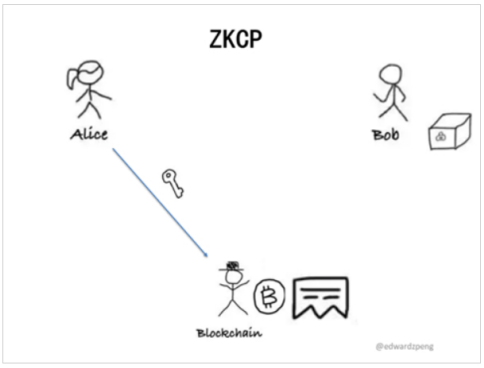
So, here is a solution. It is called zero knowledge contingent payment. ZKCP. In ZKCP, Alice first randomly choose an encryption key. Then She encryption the data she wants to sell. And she computes a zero-knowledge proof that the data satisfied the function f, which means that the data is indeed the signing key of iphone's firmware. And Alice also prove that the hash of the encryption hey is H.



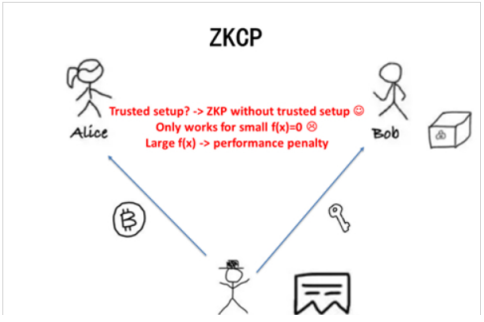
Then Alice send the encrypted data, the hash of the encryption key, and the proof to bob.

Bob will first verify the proof. To make sure Alice does not lie. And in this stage, bob cannot get the data. Because the data is encrypted and the proof is zero knowledge.

After bob make sure the proof is right. Bob setup a contract to the blockchain. The contract verifies the input of Alice is a preimage of the hash. Which means input of Alice is the encryption key of the encrypted data. If it is, then the contract will send money to Alice.



After the contract is setup in the blockchain. Alice send the encryption key to the contract.

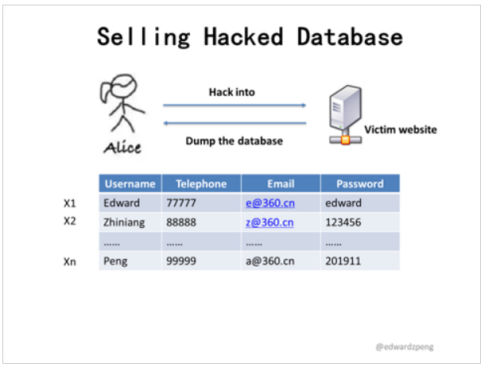


Then Alice get the money, and bob get the key to decrypt the data.

But you may there are also some weakness of this system. Because it uses ZKP, so does it need a trusted setup? Who will be the one to generate the parameter? To solve the issues, we can use an ZKP system without trusted setup. Because the proof in ZKCP only need to be verify once, so we don't need to use libSnarks for better performance. Then we can skip the setup.

Another problem is the performance. Indeed, in this system, you need to prove your data satisfy some function f(x) equal to zero. If the function f is small, the proof can work efficiently. But if the function is very complicate, then the performance will be bad.

So, can we sell large data with better performance?



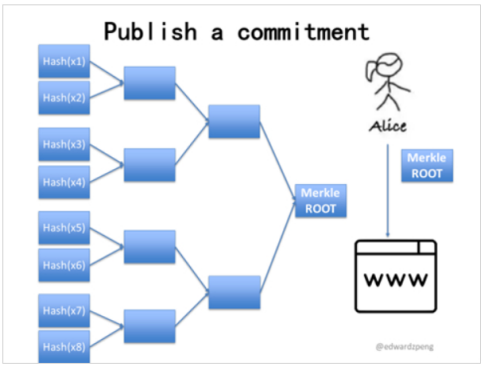
Here is an application I designed for hacker to sell a hacked database fairly.

Suppose Alice hack into a victim website, and dump the database of it. The database contains many sensitive customer information. As we can see, in this table, there are n record of customer information.

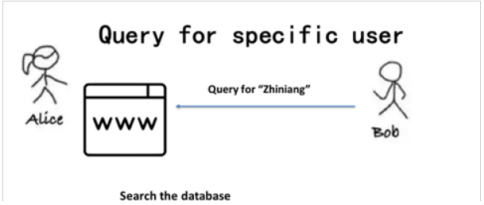
Alice wants to sell it in the dark net.

But here is problem. We don't have a function f(x) to measure the correctness of the data. Because nobody knows what exactly the data looks like. And the data maybe very large. Maybe more than 100 Gigabyte, so directly use ZKCP seems not a good idea.

How can we do that? Here is my solution, Alice can commit to the database first, then the buyer bob can buy some data to check the correctness.

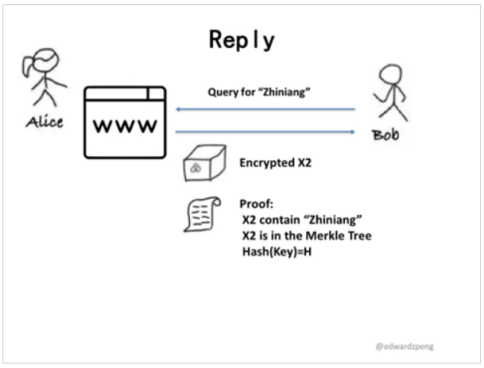


First, alice compute the hash of every record. And then, alice can build a merkle root, which is a commitment to the data. Then Alice can publish merkle root hash in the internet or blockchain.



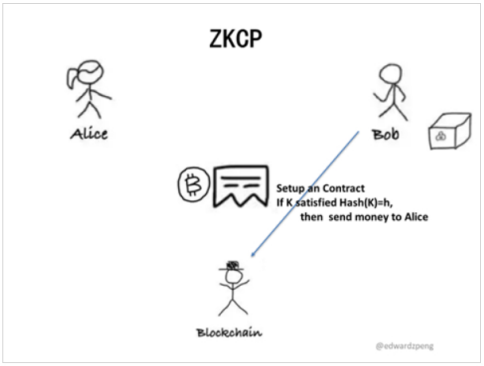
Then bob wants to check the data. He can query for some data he knows must be inside the database. For example, Bob know Zhiniang is a customer of the victim website. Then he can ask Alice to give the information of Zhiniang to him.

Alice then search the database, and he found out the X2 contain Zhiniang. Then Alice randomly chose an encryption key, and encrypt x2.

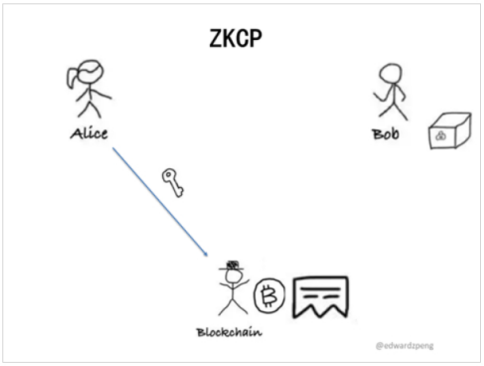


Alice send the encrypted x2 to bob. In the meantime, he also sends a zero-knowledge proof to bob. He can proof that X2 contain Zhiniang, and X2 is in the Merkle tree she published before, and the hash of the encryption key for the encrypted X2 equal to H.

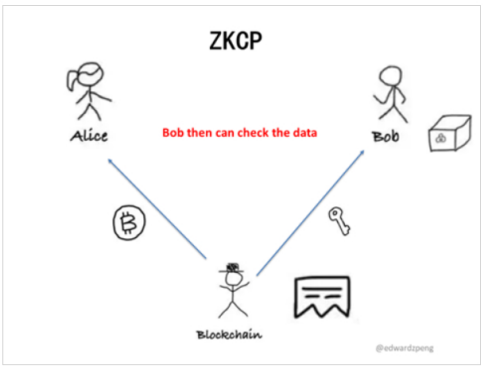
This is very efficient using ZKP.



Then bob check the proof and setup a contract to ask for the decryption key of the encrypted data.

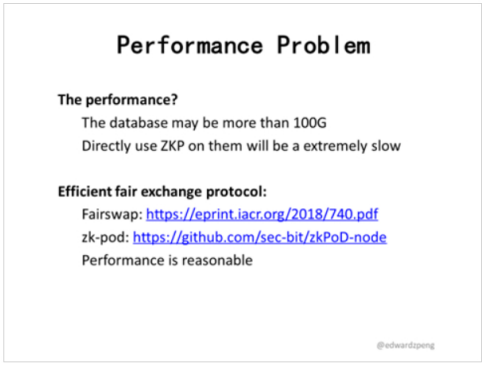


After the contract deployed, Alice send the key to the contract.



Then Alice get the money for one record. And bob get the decryption of X2.

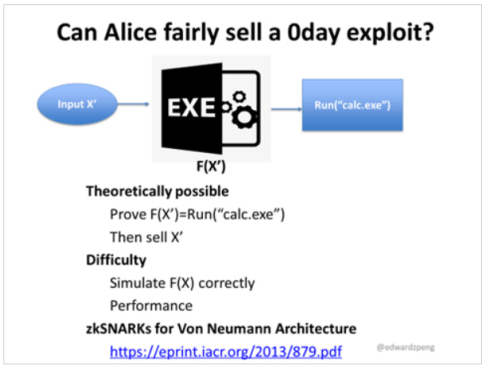
Then bob can check the data.



How about the performance of buy a lot of record from Alice? The database maybe more than 100 gigabytes, directly use ZKP on them will be extremely slow.

There are some efficient protocol doing fair exchange. For example, the Fairswap, and zk-pod. The performance is reasonable.

So, hackers. Stop selling the hacked database in the old way. Use the new technology.



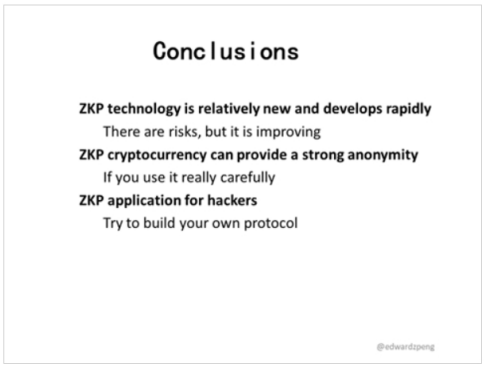
At last, can Alice fairly sell a 0day exploit like this.

My answer is, it's theoretically possible. But not practical now. We can think 0 zero day is actually an input for some program, satisfy that the result of the program is pop a calculator. Then selling a zero day is actually selling a input of function X with a particular output. So it's perfectly suitable for ZKP.

However, there are many difficulties in real application. Because it very hard to simulate F(x) correctly, and the performance is very bad for a large binary like JavaScript engine.

But it's theoretical possible now. There is some paper talk about simulate von Neumann Architecture using ZKP. You can read the paper here.

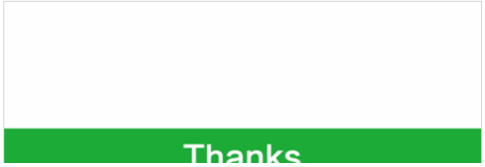
It's not practical today, but I think I will be useable in the future.

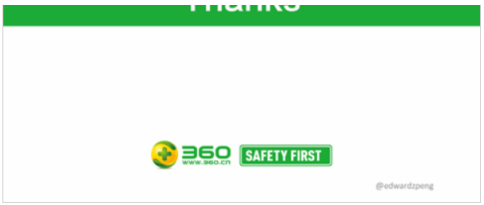


Here is the conclusion.

ZKP technology is relatively new and develops very rapidly. There are risks, But it is improving ZKP cryptocurrency can provide a strong anonymity. If you use it really carefully and really understand the security model.

There will be many new ZKP application for hackers, try to build your own protocol.





Ok, that all. Thank you for your time. Hope it's useful for you.

本文链接：https://blogs.360.cn/post/Security-Risks-in-Zero-Knowledge-Proof-Cryptocurrencies.html

-- EOF --

作者 admin001 发表于 *2019-12-20 18:07:08*，添加在分类 cryptography 下，最后修改于 *2019-12-20 19:14:23*

分享到：新浪微博微信Twitter印象笔记QQ好友有道云笔记

« 蔓灵花（APT-C-08）移动平台攻击活动揭露
Darkhotel（APT-C-06）使用"双星"0Day漏洞（CVE-2019-17026、CVE-2020-0674）针对中国发起的APT攻击分析 »

Comments