CS 5602 Introduction to Cryptography
Lecture 25
Public Key Cryptography

George Markowsky
Computer Science Department
Missouri University of Science & Technology

1

Chapter Goals

- To learn about public key encryption and the hard problems on which it is based.
- To understand the RSA algorithm and the assumptions on which its security relies.
- To understand the ElGamal encryption algorithm and it assumptions.
- To learn about the Rabin encryption algorithm and its assumptions.
- To learn about the Paillier encryption algorithm and its assumptions.

2

Public Key Cryptography

- Recall that in symmetric key cryptography each communicating party needed to have a copy of the same secret key.
- This led to a very difficult key management problem.
- In public key cryptography we replace the use of identical keys with two keys, one public and one private.
- The public key can be published in a directory along with the user's name.
- Anyone who then wishes to send a message to the holder of the associated private key will take the public key, encrypt a message under it and send it to the owner of the corresponding private key.
- The idea is that only the holder of the private key will be able to decrypt the message.
- More clearly, we have the transformations

Message + Alice's public key = Ciphertext
Ciphertext + Alice's private key = Message

3

Public Key Cryptography

- Public key systems work because the two keys are linked in a mathematical way, such that knowing the public key tells you nothing about the private key.
- But knowing the private key allows you to unlock information encrypted with the public key.
- This may seem strange, and will require some thought and patience to understand.
- The concept was so strange it was not until 1976 that anyone thought of it.
- The idea was first presented in the seminal paper of Diffie and Hellman entitled *New Directions in Cryptography*.
- Although Diffie and Hellman invented the concept of public key cryptography it was not until a year or so later that the first (and most successful) system, namely RSA, was invented.

4

Public Key Cryptography

- The previous paragraph is how the 'official' history of public key cryptography goes.
- However, in the late 1990s an unofficial history came to light.
- It turned out that in 1969, over five years before Diffie and Hellman invented public key cryptography, a cryptographer called James Ellis, working for the British government's communication headquarters GCHQ, invented the concept of public key cryptography (or non-secret encryption as he called it) as a means of solving the key distribution problem.
- Ellis, just like Diffie and Hellman, did not however have a system.

5

Public Key Cryptography

- The problem of finding such a public key encryption system was given to a new recruit to GCHQ called Clifford Cocks in 1973.
- Within a day Cocks had invented what was essentially the RSA algorithm, although a full four years before Rivest, Shamir and Adleman.
- Hence, by 1974 the British security services had already discovered the main techniques in public key cryptography.

6

Trapdoor One-Way Functions

- There is a surprisingly small number of ideas behind public key encryption algorithms, which may explain why once Diffie and Hellman or Ellis had the concept of public key encryption, an invention of essentially the same cipher, i.e. RSA, came so quickly.
- There are so few ideas because we require a mathematical operation which is easy to do one way, i.e. encryption, but which is hard to do the other way, i.e. decryption, without some special secret information, namely the private key.
- Such a mathematical function is called a trapdoor one-way function, since it is effectively a one-way function unless one knows the key to the trapdoor.

7

Trapdoor One-Way Functions

- Luckily there are a number of possible one-way functions which have been well studied, such as factoring integers, computing discrete logarithms or computing square roots modulo a composite number.
- We shall now study such one-way functions, before presenting some public key encryption algorithms
- However, these are only computational one-way functions in that given enough computing power one can invert these functions faster than exhaustive search.

8

Candidate One-Way Functions

- The most important one-way function used in public key cryptography is that of factoring integers.
- By factoring an integer we mean finding its prime factors, for example

$$10 = 2 \cdot 5$$
$$60 = 2^2 \cdot 3 \cdot 5$$
$$2^{113} - 1 = 3391 \cdot 23\,279 \cdot 65\,993 \cdot 1\,868\,569 \cdot 1\,066\,818\,132\,868\,207$$

- Finding the factors is an expensive computational operation.

9

$L_N(\alpha, \beta) = \exp((\beta + o(1))(\log N)^{\alpha}(\log \log N)^{1-\alpha})$

- Quite an interesting function – let's consider some values
- $O(L_N(0, \beta)) = \exp((\beta + o(1))(\log \log N)) = \exp(\log \log N)^{(\beta + o(1))} = (\log N)^{(\beta + o(1))}$ which is polynomial in log N, the size of the problem
- $O(L_N(1, \beta)) = (\log N)^{(\log N)(\beta + o(1))}$ which is exponential in log N
- Hence, the function LN (α, β)
- for $0 < \alpha < 1$ interpolates between polynomial and exponential time.
- An algorithm with complexity $O(L_N(\alpha, \beta))$ for $0 < \alpha < 1$ is said to have sub-exponential behavior.
- Notice that multiplication, which is the inverse algorithm to factoring, is a very simple operation requiring time less than O(LN (0, 2)).

10

Factoring Methods

- Trial Division: Try every prime number up to $\sqrt{N}$ and see if it is a factor of N.
- This has complexity $L_N(1, 1)$, and is therefore an exponential algorithm.
- Elliptic Curve Method: This is a very good method if $p < 2^{50}$, its complexity is
- $L_p(1/2, c)$, which is sub-exponential.
- Notice that the complexity is given in terms of the size of the unknown value p.
- If the number is a product of two primes of very unequal size then the elliptic curve method may be the best at finding the factors.
- Quadratic Sieve: This is probably the fastest method for factoring integers of between 80 and 100 decimal digits. It has complexity $L_N(1/2, 1)$.
- Number Field Sieve: This is currently the most successful method for numbers with more than 100 decimal digits.
- It can factor numbers of the size of $10^{155} \approx 2^{512}$ and has complexity $L_N(1/3, 1.923)$.

11

Other Hard Problems Related to Factoring

- There are a number of other hard problems related to factoring which can be used to produce public key cryptosystems.
- Suppose you are given N but not its factors p and q, there are four main problems which one can try to solve:

1. FACTORING: Find p and q.
2. RSA: Given e such that $GCD(e, (p - 1)(q - 1)) = 1$ and c, find m such that $m^e = c \pmod{N}$.
3. QUADRES: Given a, determine whether a is a square modulo N.
4. SQRROOT: Given a such that $a = x^2 \pmod{N}$, find x.

12

Discrete Logarithm Problems

- Another important class of problems are those based on the discrete logarithm problem or its variants.
- Let (G, ·) be a finite abelian group, such as the multiplicative group of a finite field or the set of points on an elliptic curve over a finite field.
- The discrete logarithm problem, or DLP, in G is given g, h ∈ G, find an integer x (if it exists) such that $g^x = h$.
- For some groups G this problem is easy.
- For example if we take G to be the integers modulo a number N under addition then given g, h ∈ $\mathbb{Z}/N\mathbb{Z}$ we need to solve x · g = h.

13

Discrete Logarithm Problems

- We have already seen in Chapter 1 that we can easily tell whether such an equation has a solution, and determine its solution when it does, using the extended Euclidean algorithm.
- For certain other groups determining discrete logarithms is believed to be hard.
- For example in the multiplicative group of a finite field the best known algorithm for this task is the Number Field Sieve.
- The complexity of determining discrete logarithms in this case is given by L_N (1/3, c) for some constant c, depending on the type of the finite field, e.g. whether it is a large prime field or an extension field of characteristic two.

14

Discrete Logarithm Problems

- For other groups, such as elliptic curve groups, the discrete logarithm problem is believed to be even harder.
- The best known algorithm for finding discrete logarithms on a general elliptic curve defined over a finite field F_q is Pollard's Rho method which has complexity $\sqrt{q} = L_q(1, 1/2)$
- Hence, this is a fully exponential algorithm.
- Since determining elliptic curve discrete logarithms is harder than in the case of multiplicative groups of finite fields we are able to use smaller groups.
- This leads to an advantage in key size.
- Elliptic curve cryptosystems often have much smaller key sizes (say 160 bits) compared with those based on factoring or discrete logarithms in finite fields (where for both the 'equivalent' recommended key size is about 1024 bits).

15

Discrete Logarithm Problems

- Just as with the FACTORING problem, there are a number of related problems associated to discrete logarithms; again suppose we are given a finite abelian group (G, ·) and g ∈ G.
- DLP: This is the discrete logarithm problem considered above. Namely given g, h ∈ G such that $h = g^x$, find x.
- DHP: This is the Diffie–Hellman problem. Given g ∈ G and $a = g^x$ and $b = g^y$, find c such that $c = g^{xy}$.
- DDH: This is the decision Diffie–Hellman problem. Given g ∈ G and $a = g^x$, $b = g^y$ and $c = g^z$, determine if z = x · y.

16

Problem Reductions

- Lemma 11.1. In an arbitrary finite abelian group G the DHP is no harder than the DLP.
- Proof: Suppose there is an oracle O_{DLP} that will solve the DLP, i.e. on input of $h = g^x$ it will return x.
- To solve the DHP on input of $a = g^x$ and $b = g^y$ we compute

1. $x = O_{DLP}(a)$.
2. $c = b^x$.
3. Output c.

- The above reduction clearly runs in polynomial time and will compute the true solution to the DHP, assuming the oracle returns the correct value, i.e.
- Hence, the DHP is no harder than the DLP.

17

Problem Reductions

- Lemma 11.2. In an arbitrary finite abelian group G the DDH is no harder than the DHP.
- Lemma 11.3. The FACTORING and SQRROOT problems are polynomial-time equivalent.
- Lemma 11.4. The RSA problem is no harder than the FACTORING problem.
- There is some evidence, although slight, that the RSA problem may actually be easier than FACTORING for some problem instances.
- It is a major open question as to how much easier it is.

18

G.H. Hardy

- In *The Mathematician's Apology*
  - *Real mathematics has no effects on war. No one has yet discovered any warlike purpose to be served by the theory of numbers.*
- Wars of the future will be information wars in part

19

RSA Public Key System

- Pick very large primes P and Q
- Let R = P*Q
- Let F = (P-1)*(Q-1)
- Find Y with GCD(F,Y) = 1
- Find A and B with A*Y+B*F = 1
- Publish R and Y
- To encode M (message viewed as a large number) compute
  - E = M^Y mod R
- To decode, compute
  - E^A mod R

20

Why Does RSA Work?

E^A mod R =

M^{AY} mod R =

M^{1-BF} mod R =

$M*(M^F)^{-B}$ mod R=

$M*1^{-B}$ mod R=

M mod R

If anyone figures out how to factor efficiently, we will have a BIG problem on our hands!

21

RSA Public Key System

- Pick very large primes P and Q
- Let R = P*Q
- Let F = (P-1)*(Q-1)
- Note that F = ϕ(R)
- Find Y with GCD(F,Y) = 1
- It is common to choose Y = 3, 17, or 65,537
- Find A and B with A*Y+B*F = 1
- Publish R and Y
- To encode M (message viewed as a large number) compute
  - E = M^Y mod R
- To decode, compute
  - E^A mod R

22

RSA Encryption

- r = pq (p and q are large primes)
- We will look at ($\mathbb{Z}_r$,×), let's call it $\mathbb{Z}_r$*
- Since r is not prime we are looking at the multiplicative group of units
- |$\mathbb{Z}_r$*| = φ(r) = (p-1)(q-1) = f (in the original scheme)
- Pick y coprime with f and find a and b such that
- ay + bf = 1
- Let m be our message encoded (not encrypted) as a number
- Let the encrypted message e be computed as m^a (mod r)
- That's the encryption!

23

RSA Decryption

- d = e^a (mod r)
- That's the decryption!
- Why does this work?
- e^a (mod r) = m^{ay} (mod r)
- If m ϵ $\mathbb{Z}_r$* we know that $m^f = m^{\varphi(r)} = 1$ (mod r), so
- $m^{ay} = m^{1-bf} = mm^{-bf} = m(m^{-b})^f = m$ (mod r) since $k^f = 1$ for all k ϵ $\mathbb{Z}_r$*
- That's it folks! An idea worth 100's of millions of $
- There is a slight glitch in the above presentation – can you see it?
- What if m ∉ $\mathbb{Z}_r$*?

24

Security of RSA

- Lemma 11.5. If the RSA problem is hard then the RSA system is secure under a chosen plaintext attack, in the sense that an attacker is unable to recover the whole plaintext given the ciphertext.
- Lemma 11.6. If one knows the RSA decryption exponent d corresponding to the public key (N, e) then one can efficiently factor N.
- Lemma 11.7. Given an RSA modulus N and the value of $\Phi = \phi(N)$ one can efficiently factor N.

25

Security of RSA

- Use of a Shared Modulus. Since modular arithmetic is very expensive it can be very tempting for a system to be set up in which a number of users share the same public modulus N but use different public/private exponents, (e_i, d_i).
- One reason to do this could be to allow very fast hardware acceleration of modular arithmetic, specially tuned to the chosen shared modulus N.
- This is, however, a very silly idea since it can be attacked in one of two ways, either by a malicious insider or by an external attacker.
- DON'T DO IT!

26

Security of RSA

- Use of a Small Public Exponent.
- Fielded RSA systems often use a small public exponent e so as to cut down the computational cost of the sender.
- We shall now show that this can also lead to problems.
- Suppose we have three users all with different public moduli N1, N2, and N3.
- In addition suppose they all have the same small public exponent e = 3.
- Can break this system if get them all to transmit the same message.
- DON'T DO IT!

27

Security of RSA

- The main lesson, however, from both these attacks is that plaintext should be randomly padded before transmission.
- That way the same 'message' is never encrypted to two different people.
- In addition, one should probably avoid very small exponents for encryption, e = 65 537 is the usual choice now in use.
- However, small public exponents for RSA signatures do not seem to have any problems.

28

ElGamal Encryption

- The simplest encryption algorithm based on the discrete logarithm problem is the ElGamal encryption algorithm.
- Unlike the RSA algorithm, in ElGamal encryption there are some public parameters that can be shared by a number of users.
- These are called the domain parameters and are given by:
- p a 'large prime', by which we mean one with around 1024 bits, such that p−1 is divisible by another 'medium prime' q of around 160 bits.
- g an element of F_p^* of prime order q, i.e. $g = r^{(p-1)/q} \pmod p \neq 1$ for some $r \in F_p^*$.

29

ElGamal Encryption

- All the domain parameters do is create a public finite abelian group G of prime order q with generator g.
- Such domain parameters can be shared between a large number of users.
- Once these domain parameters have been fixed, the public and private keys can then be determined.
- The private key is chosen to be an integer x, whilst the public key is given by
- $h = g^x \pmod p$.
- Notice that whilst each user in RSA needed to generate two large primes to set up their key pair (which is a costly task), for ElGamal encryption each user only needs to generate a random number and perform a modular exponentiation to generate a key pair.

30

ElGamal Encryption

- Messages are assumed to be non-zero elements of the field $F_p{}^*$. To encrypt a message $m \in F_p{}^*$ we
1. generate a random ephemeral key k,
2. set $c_1 = g^k$,
3. set $c_2 = m \cdot h^k$,
4. output the ciphertext as $c = (c_1, c_2)$.
- Notice that since each message has a different ephemeral key, encrypting the same message twice will produce different ciphertexts.

31

ElGamal Encryption

- To decrypt a ciphertext $c = (c_1, c_2)$ we compute
- $c_2 / c_1{}^x = m \cdot h^k/g^{xk} = m \cdot g^{xk}/g^{xk} = m$.
- Lemma 11.8. Assuming the Diffie–Hellman problem (DHP) is hard then ElGamal is secure under a chosen plaintext attack, where security means it is hard for the adversary, given the ciphertext, to recover the whole of the plaintext.

32

Rabin Encryption

- There is another system, due to Rabin, based on the difficulty of factoring large integers.
- In fact it is actually based on the difficulty of extracting square roots modulo $N = p \cdot q$.
- Recall that these two problems are known to be equivalent, i.e.,
1. knowing the factors of N means we can extract square roots modulo N,
2. extracting square roots modulo N means we can factor N.
- Hence, in some respects such a system should be considered more secure than RSA.
- Encryption in the Rabin encryption system is also much faster than almost any other public key scheme.
- Despite these plus points the Rabin system is not used as much as the RSA system.
- It is, however, useful to study for a number of reasons, both historical and theoretical.
- The basic idea of the system is also used in some higher level protocols.

33

Rabin Encryption

- We first choose prime numbers of the form
- $p \equiv q \equiv 3 \pmod 4$
- since this makes extracting square roots modulo p and q very fast.
- The private key is then the pair (p, q). To compute the associated public key we generate a random integer $B \in \{0, \ldots, N-1\}$ and then the public key is (N, B), where N is the product of p and q.
- To encrypt a message m, using the above public key, in the Rabin encryption algorithm we compute
- $c = m(m + B) \pmod N$.
- Hence, encryption involves one addition and one multiplication modulo N.
- Encryption is therefore much faster than RSA encryption, even when one chooses a small RSA encryption exponent.

34

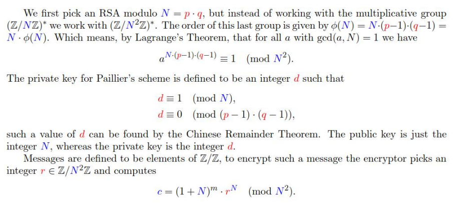
Rabin Decryption

- Decryption is far more complicated, essentially we want to compute
- $m = \sqrt{\frac{B^2}{4} + c} - \frac{B}{2}$ (mod N) this is the quadratic formula for $m^2 + Bm - c = 0$
- At first sight this uses no private information, but a moment's thought reveals that you need the factorization of N to be able to find the square root.
- There are however four possible square roots modulo N, since N is the product of two primes.
- Hence, on decryption you obtain four possible plaintexts.
- This means that we need to add redundancy to the plaintext before encryption in order to decide which of the four possible plaintexts corresponds to the intended one.

35

Paillier Encryption

- There is another system, due to Paillier, based on the difficulty of factoring large integers.
- Paillier's scheme has a number of interesting properties, such as the fact that it is additively homomorphic (which means it has found application in electronic voting applications).



We first pick an RSA modulo $N = p \cdot q$, but instead of working with the multiplicative group $(\mathbb{Z}/N\mathbb{Z})^*$ we work with $(\mathbb{Z}/N^2\mathbb{Z})^*$. The order of this last group is given by $\phi(N) = N \cdot (p-1) \cdot (q-1) = N \cdot \phi(N)$. Which means, by Lagrange's Theorem, that for all a with $\gcd(a, N) = 1$ we have

$$a^{N\cdot(p-1)\cdot(q-1)} \equiv 1 \pmod{N^2}.$$

The private key for Paillier's scheme is defined to be an integer d such that

$$d \equiv 1 \pmod N,$$
$$d \equiv 0 \pmod{(p-1)\cdot(q-1)},$$

such a value of d can be found by the Chinese Remainder Theorem. The public key is just the integer N, whereas the private key is the integer d.

Messages are defined to be elements of $\mathbb{Z}/\mathbb{Z}$, to encrypt such a message the encryptor picks an integer $r \in \mathbb{Z}/N^2\mathbb{Z}$ and computes

$$c = (1 + N)^m \cdot r^N \pmod{N^2}.$$

36

Paillier Encryption

To decrypt one first computes

$$\begin{aligned} t &= c^d \pmod{N^2} \\ &= (1+N)^{m\cdot d} \cdot r^{d\cdot N} \pmod{N^2} \\ &= (1+N)^{m\cdot d} \pmod{N^2} \text{ since } d \equiv 0 \pmod{(p-1)\cdot(q-1)} \\ &= 1 + m\cdot d\cdot N \pmod{N^2} \\ &= 1 + m\cdot N \pmod{N^2} \text{ since } d \equiv 1 \pmod{N}. \end{aligned}$$

Then to recover the message we compute

$$R = \frac{t-1}{N}.$$

37

New Applications of Hashing and Public Key Cryptography

- Cryptocurrencies – Bitcoin, etc.
- How do they work?
- Will not go into lots of details, but focus on the role that hashing and public keys play
- The basic concept is that of a blockchain
- Let's give a quick overview of some of the ideas

38

Satoshi Nakamoto (Wikipedia)

- Satoshi Nakamoto is the name used by the unknown person or people who designed bitcoin and created its original reference implementation.
- As part of the implementation, they also devised the first blockchain database.
- In the process they were the first to solve the double-spending problem for digital currency.
- They were active in the development of bitcoin up until December 2010.

39

Satoshi Nakamoto (Wikipedia)

- In October 2008, Nakamoto published a paper on the cryptography mailing list at metzdowd.com describing the bitcoin digital currency.
- It was titled "Bitcoin: A Peer-to-Peer Electronic Cash System".
- In January 2009, Nakamoto released the first bitcoin software that launched the network and the first units of the bitcoin cryptocurrency, called bitcoins.
- Satoshi Nakamoto released the Version 0.1 of bitcoin software on Sourceforge on 9 January 2009.

40

Satoshi Nakamoto (Wikipedia)

- As the sole, predominant early miner, the inventor was awarded bitcoin at genesis and for 10 days afterwards.
- Except for test transactions these remain unspent since mid-January 2009.
- The public bitcoin transaction log shows that Nakamoto's known addresses contain roughly one million bitcoins.
- As of 17 December 2017, this is worth over 19 billion USD.
- This makes him the 44th richest person on Earth.
- Lately, there has been a Satoshi Nakamoto controversy and a lot of effort to identify him/her

41

Craig Steven Wright (Wikipedia)

- Craig Steven Wright (born October 1970) is an Australian computer scientist and businessman.
- He has publicly claimed to be the main part of the team that created bitcoin, and the identity behind the pseudonym Satoshi Nakamoto.
- These claims are widely regarded as a possible hoax.
- The dispute between Wright and his attackers has gotten very acrimonious and is now being decided in part through the court system
- I suspect that by the end of 2019 we will know who Satoshi Nakamoto is

42

CCN LIVE MARKET CRYPTO

Home ▸ Bitcoin Law ▸ Satoshi Nakamoto Horde: Craig Wright Submits List of Bitcoin Addresses

Craig Wright's lawyer has furnished a list of bitcoin addresses that would presumably prove whether or not the Australian entrepreneur is Satoshi Nakamoto. | Source: (i) YouTube/BBC (ii) Shutterstock ; Edited by CCN

Satoshi Nakamoto Horde: Craig Wright Submits List of Bitcoin Addresses

43

Bitcoin: A Peer-to-Peer Electronic Cash System

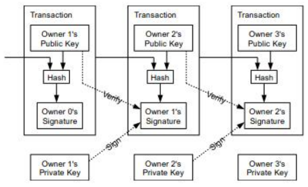
Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

44

2. Transactions

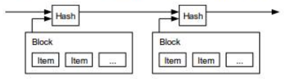
We define an electronic coin as a chain of digital signatures. Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin. A payee can verify the signatures to verify the chain of ownership.



45

3. Timestamp Server

The solution we propose begins with a timestamp server. A timestamp server works by taking a hash of a block of items to be timestamped and widely publishing the hash, such as in a newspaper or Usenet post [2-5]. The timestamp proves that the data must have existed at the time, obviously, in order to get into the hash. Each timestamp includes the previous timestamp in its hash, forming a chain, with each additional timestamp reinforcing the ones before it.

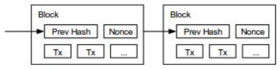


46

4. Proof-of-Work

To implement a distributed timestamp server on a peer-to-peer basis, we will need to use a proof-of-work system similar to Adam Back's Hashcash [6], rather than newspaper or Usenet posts. The proof-of-work involves scanning for a value that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash.

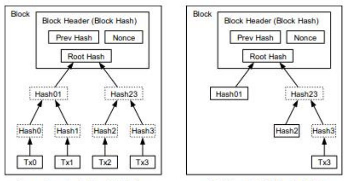
For our timestamp network, we implement the proof-of-work by incrementing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it.



47

7. Reclaiming Disk Space

Once the latest transaction in a coin is buried under enough blocks, the spent transactions before it can be discarded to save disk space. To facilitate this without breaking the block's hash, transactions are hashed in a Merkle Tree [7][2][5], with only the root included in the block's hash. Old blocks can then be compacted by stubbing off branches of the tree. The interior hashes do not need to be stored.

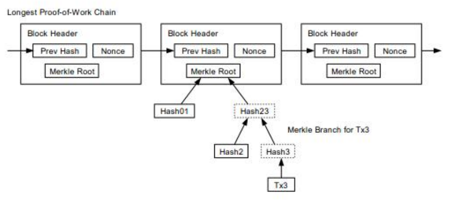


Transactions Hashed in a Merkle Tree After Pruning Tx0-2 from the Block

48

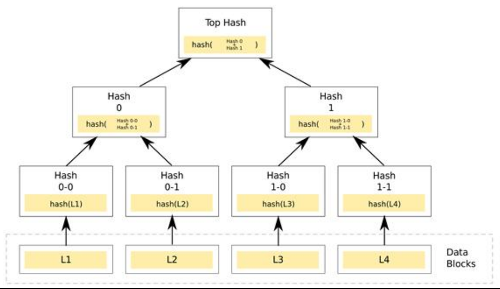
8. Simplified Payment Verification

It is possible to verify payments without running a full network node. A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest chain, and obtain the Merkle branch linking the transaction to the block it's timestamped in. He can't check the transaction for himself, but by linking it to a place in the chain, he can see that a network node has accepted it, and blocks added after it further confirm the network has accepted it.



49

Merkle Tree



50