

Bitcoin Key Management: HD Wallets, BIP032



Nicolas T. Courtois

BlockChain



Bitcoin Blockchain

Contains Digital Signatures for all transactions.

Moneys transmitted to some
 $H(\text{Public Key})$ on 160 bits exactly.

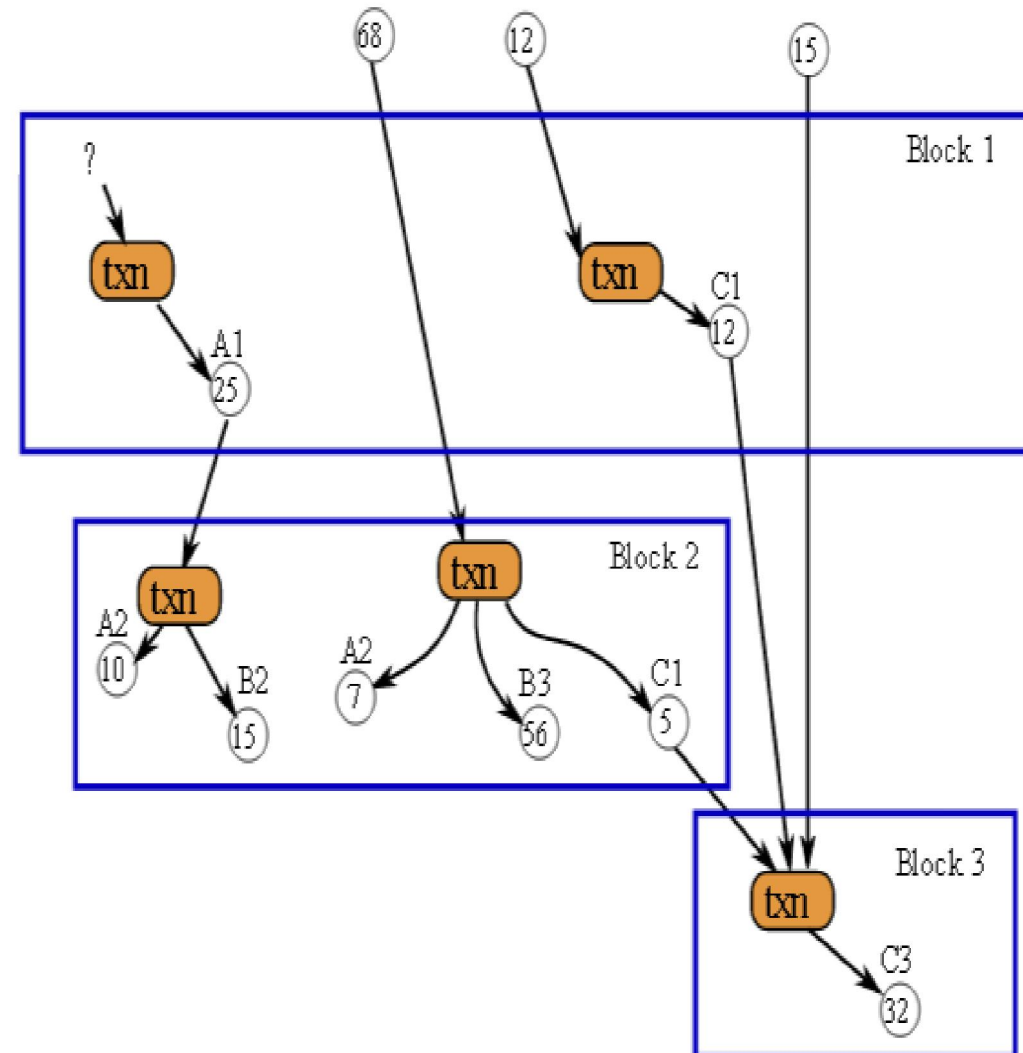
Public key is NOT revealed in the process (90%)

– fact of enormous importance,

- this is how Satoshi “dodged” some future attacks!
- there are unhappy exceptions: coinbase transactions usually are of type P2PK

Bitcoin BlockChain

Here transactions are
“written in stone”.

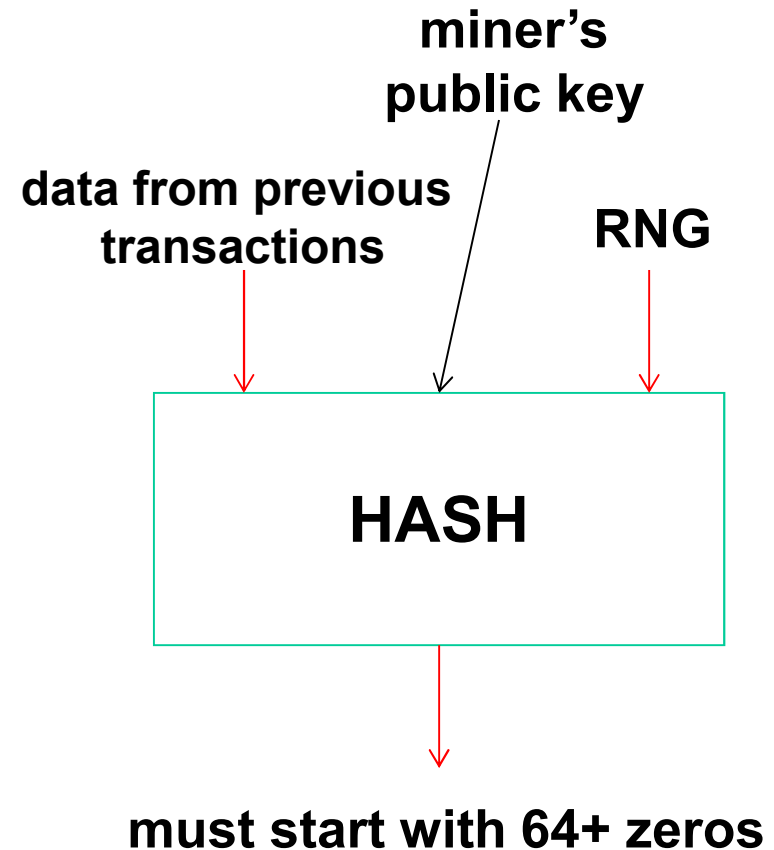


Bitcoin Mining

- Minting: creation of new currency.
- Confirmation+re-confirmation of older transactions

Ownership:

- “policed by majority of miners”:
- only the owner can transfer [a part of] 25 BTC produced.



Good Practice:

- use each ECDSA public key only once!
 - solid protection against bitcoin elliptic curve being broken
 - it will be broken in a 5-10 years from today (my prediction)
 - bitcoin allows to protect your **Public Key**
 - very surprising, security engineering, genius of Satoshi
 - your **PK** should **NOT** be public,
 - OK for keys no longer used
 - very strong protection:
 - » even if the NSA can break secp256k1, they cannot crack it in 1 second...
 - » also improves privacy [but not a panacea]

*Further Good Practice:

1. use each ECDSA public key only once!

– IN ADDITION:

2. store large amounts into many accounts

(tip: use standardized amounts)

– WHY?

– a) smaller targets,

- cost > profit for each attack...
- less visible targets

– b) weak keys,

- if we move 1M BTC all the time from one public key to another, eventually one key could be weak...
- or “bad random” will be used which will betray this one key when used...

Consequences:

1. use each ECDSA public key only once!

⇒ Key management becomes a problem.

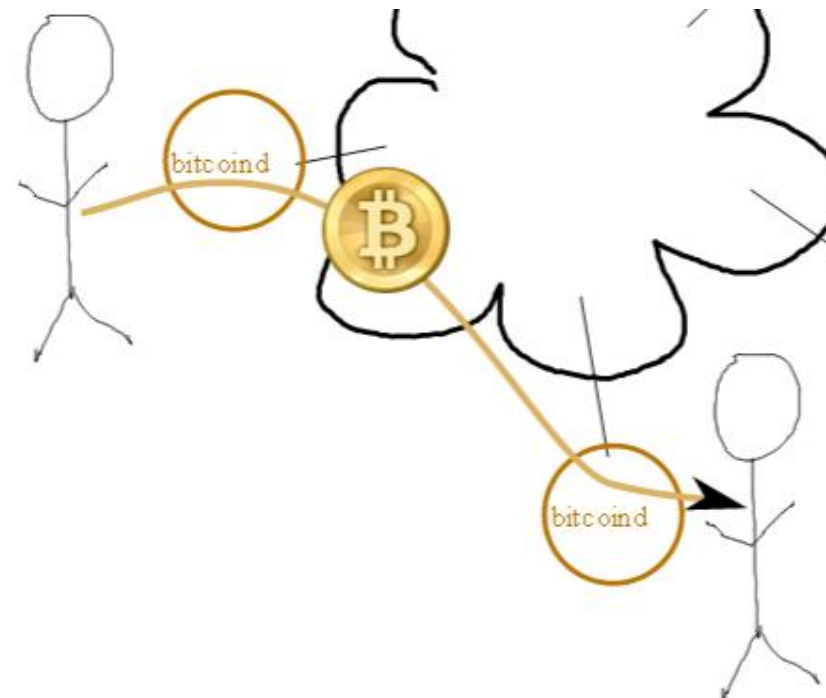
⇒ Old backups do not give access to all funds.

⇒ Etc...

A terrific new market for **managing** and **protecting** cryptographic keys is emerging.

Bitcoin Network and Transfer

To: 1K2CcfWYW5sBL2xSeQWxpcmjPCgoXdi36
 Amount: 1.0 BTC

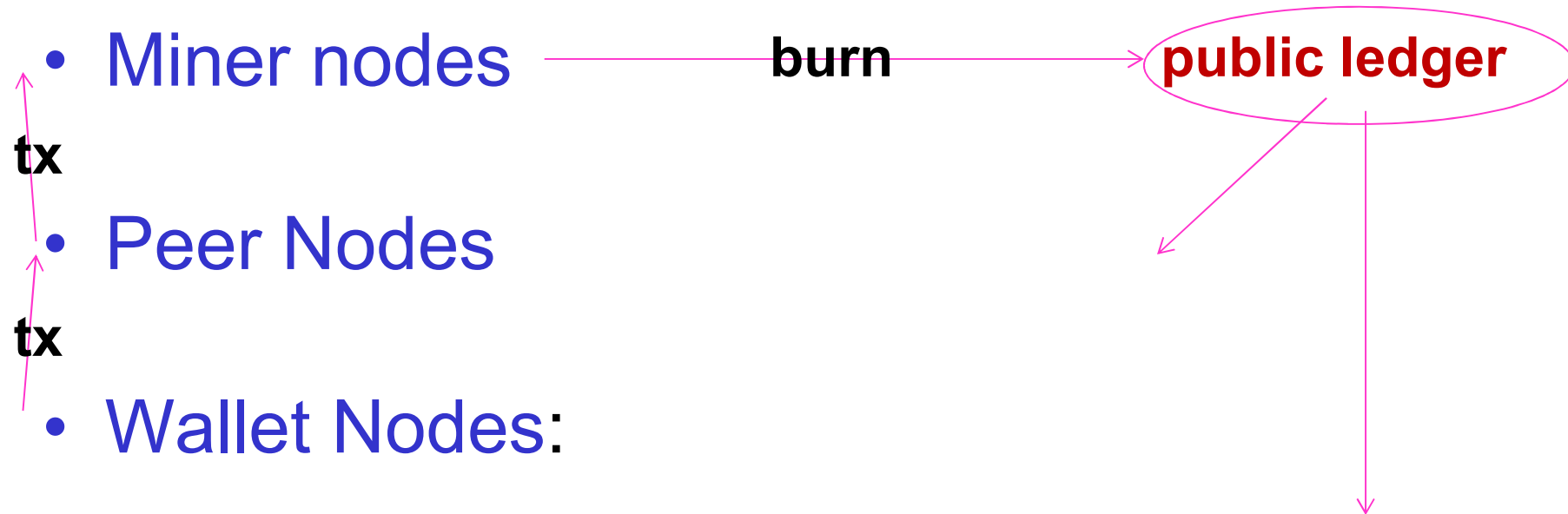


Separation?

It is possible to almost totally separate:

- **Miner nodes**
 - Hashing with public keys
- **Peer Nodes**
 - Relay and store transactions and blocks
- **Wallet Nodes:**
 - Store and release funds,
 - Focus on management of private keys, master keys etc..

Tx LifeCycle



*Why Separation

Separation makes a lot of sense in security engineering:

Principle of **Separation of Privileges** [Saltzer and Schroeder 1975]

Split software into pieces with limited privileges!

Very common in software engineering [but not yet in Bitcoin]:

- creating a child process,
- dropping privileges, etc.

Advantages:

- ⇒ A successful exploit against the larger program will gain minimal access.
- ⇒ Different people can have access to these applications
- ⇒ Pieces can run on distinct hardware (+different OS etc)!!!!!!
- ⇒ Wallets do NOT need to be connected to the Internet (or only 1 way).



*Panic – May 2014

- # active nodes << #miners
- 5K << 100K

www.coindesk.com/bitcoin-nodes-need/

Waning support

Looking at a 60-day chart of bitcoin nodes shows that the number has gone down significantly. It went from 10,000 reachable nodes in early March to below 8,000 at the beginning of May.



Source: Bitnodes

Wallets

- **Wallet**: file which stores your “money”.
- A Bitcoin client App
is also called **a wallet**



Existing Wallets

bitcoin.org/en/choose-your-wallet

Future of Wallets

Further specialization...

In Practice



Full P2P Client

<http://bitcoin.org/en/download>

Download Bitcoin-Qt

Latest version: 0.8.6 



Download Bitcoin-Qt



You will need to be patient

24 giga, 48 hours...



Windows (exe)

~12MB



Mac OS X

~14MB



Windows (zip)

~16MB



Linux (tgz)

~16MB



Ubuntu (PPA)

~4MB




Source code

(GitHub)

Mobile Apps - Android





Bitcoin

SEND COINS








ADDRESS BOOK

BTC1.1163

= EUR55.7050

Your Bitcoin Address:

1KGe NiDw zH5N
rdwN ETj3 hQEx
wr5H MN9e Fw

			Received	Both	Sent
	balance	67.9065	<div>  <div>Apr 6 ← 1719Pmohr5CkidX6mQ9zYj4nTPnGD</div> </div>		
CNY	rate	416.78			
	balance	465.2653	<div>  <div>Apr 5 ← Beer with Lisa</div> </div>		
DKK	rate	328.56	<div>  <div>Apr 5 → 1Q4H8CY4FpnJ93SPbdz4Cqgv714KX</div> </div>		
	balance	366.7824	<div>  <div>Apr 4 → Burger @ room77</div> </div>		
EUR (default)	rate	49.90			
	balance	55.7050	<div>  <div>Apr 4 ← 1G9Hjz1JCUqnHNQMpxLhsVL6FD8Co</div> </div>		
GBP	rate	40.74	<div>  <div>Apr 4 ← Donation</div> </div>		
	balance	45.4794			
HKD	rate	506.94	<div>  <div>Apr 3 ← 1FUGqeguKnVFavXYqKwYB7g4YKXJ4</div> </div>		

Keys Trust and Infrastructure



New Infrastructure?

We have:

- “Symmetric Key Infrastructure”:
 - like Kerberos [Unix, Windows]
- PKI = Public Key Infrastructure:
 - like in TLS [current e-commerce]
 - EU digital signature directive... SWIFT...
- “Keyless Infrastructure”:
 - provides authenticity/certification services without secret keys!
 - e.g. public append-only log such as bitcoin blockchain

=> All the 3 sorts of solutions can be combined BTW...





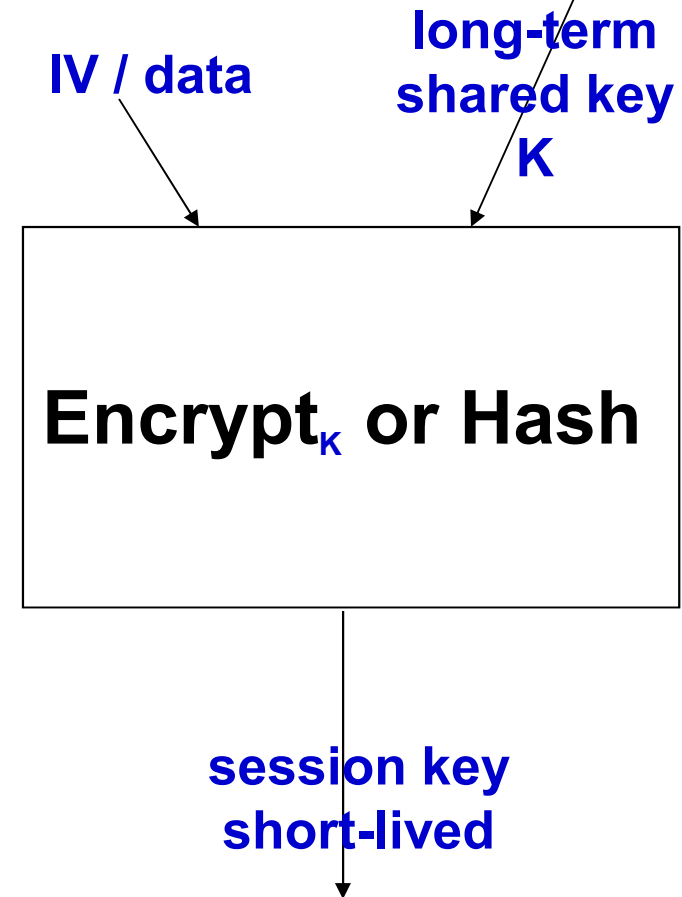
Master Keys

Common practice in the industry.

Avoids large databases
and update issues.

On this slide: how it is done
in symmetric cryptography.

MUCH more complicated with PK
crypto like in bitcoin.

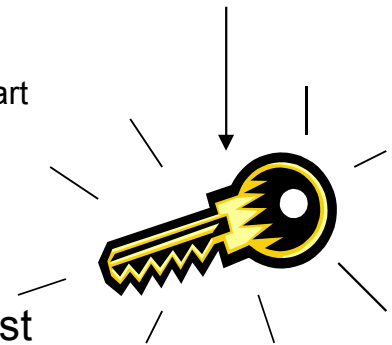




Engineering Tricks - Symmetric

Why in symmetric crypto we use **session keys** (short term keys):

- **limit key exposure**,
 - in many systems (e.g. GSM) session keys are **pre-computed in advance** by a more secure part of the system (!)
 - bank card master key never used with data chosen by the user (foil DPA)
 - forward security: cannot decrypt past communications if keys compromised
 - they has 6 master keys...
- keys should be **fresh** in order to prevent replay of the messages from the last session => total **session independence**
- **cryptanalysis**: security of symmetric crypto degrades with usage,
=> limit amount of data that the attacker can dispose of.
- better to re-establish keys when needed,
 - avoid expensive storage of too many keys locally
- in PK case, it is in fact **TOTALLY** impossible to use PK crypto to encrypt quantities of data, just too **slow**, so a symmetric key is always needed.
 - one method to get it is key establishment, studied here later.
 - second method is called hybrid encryption, e.g. in PGP / GNU PG.



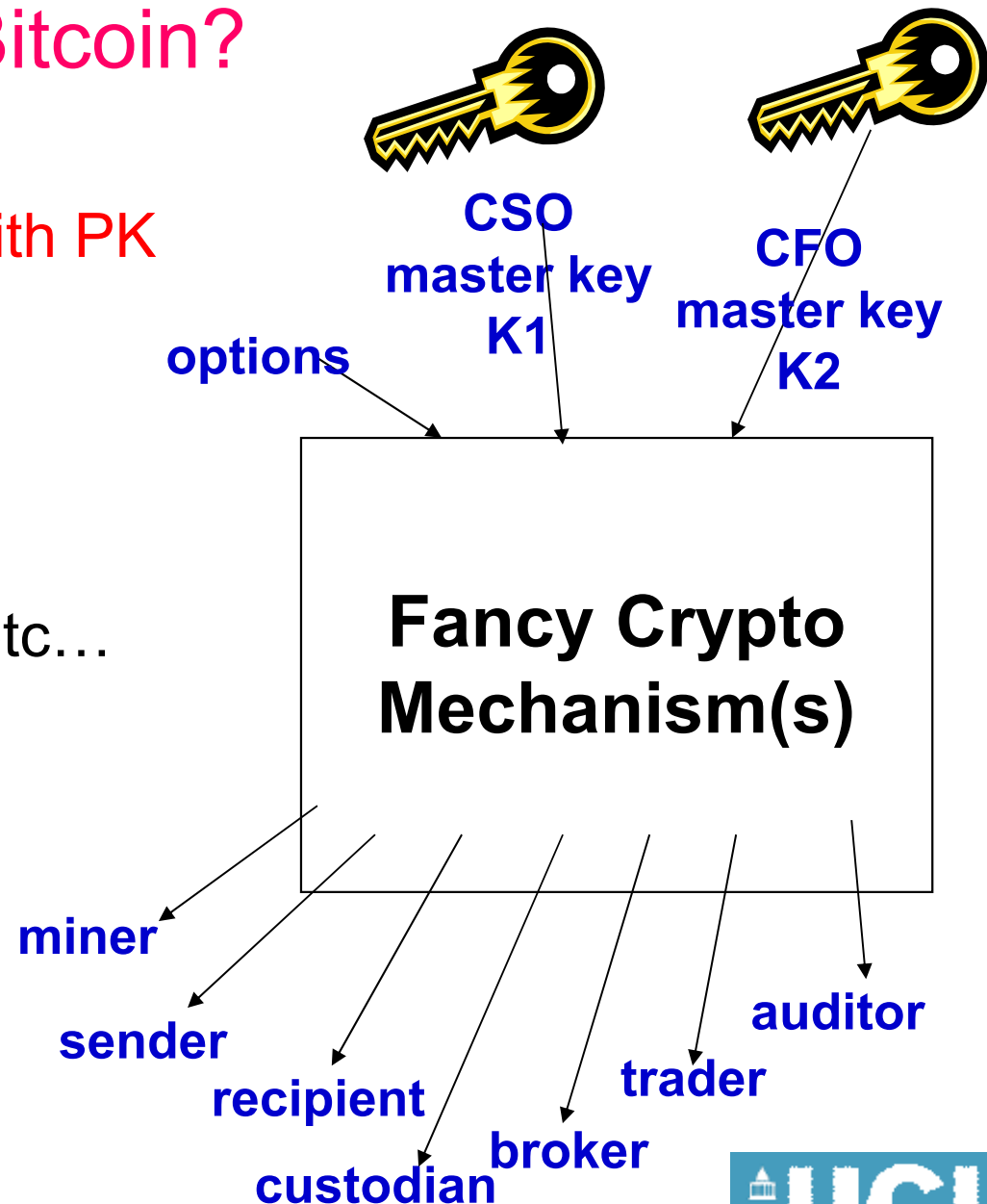
Bitcoin?

Again:

MUCH more complicated with PK
crypto like in bitcoin.

Could implement arbitrary
structures with **hierarchy**
of privileges tasks roles etc...

NOT EASY



*Warning

Possibly still NOT ideal (too centralized).

There could be other solutions
without permanent master privileges
which are a single point of failure.

Possible helpers:

Proof of stake, proof of work, smart cards etc...

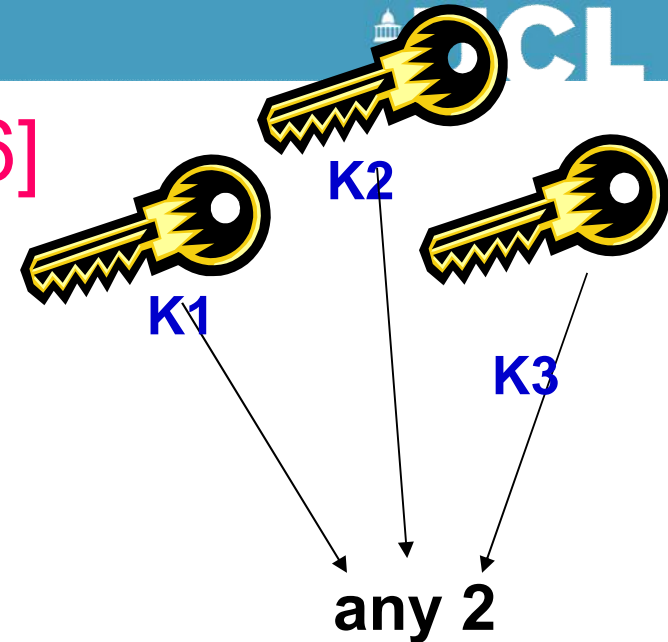


master key(s)



**Fancy Crypto
Mechanism(s)**

2outof3 [BIP16]



Bottom line:

Multisig are **MUCH stronger** than any
secret sharing
and any secret derivation mechanisms
and any HD wallets etc ...

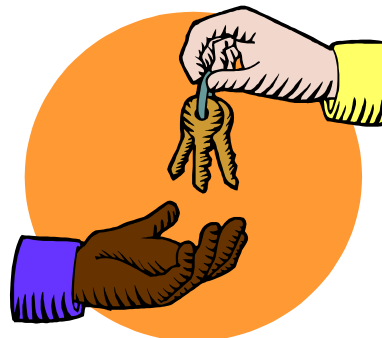
Current limitation 2outof3, will be out of 15 by the end of 2014?

Actually up to 8 out 15 already work, more or less, there are some
risks of not being able to spend later and losing money...

HD Wallets

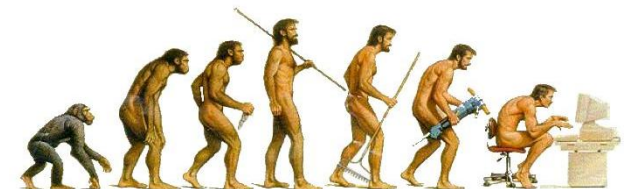
Cf. Pieter Wuille video at Bitcoin 2013 Conference by Bitcoin Foundation, San Jose, CA, May 2013

<https://www.youtube.com/watch?v=WcnMjkc31Fs>



3 Types:

0. Use random keys=> Pb. with frequent backups.
1. Type 1 = Deterministic
2. Type 2 = Deterministic + “Magic” Audit Capability
3. HD Wallets = Hierarchical Deterministic
(a tree with several levels)
 - BIP032: current standard with 4+ levels



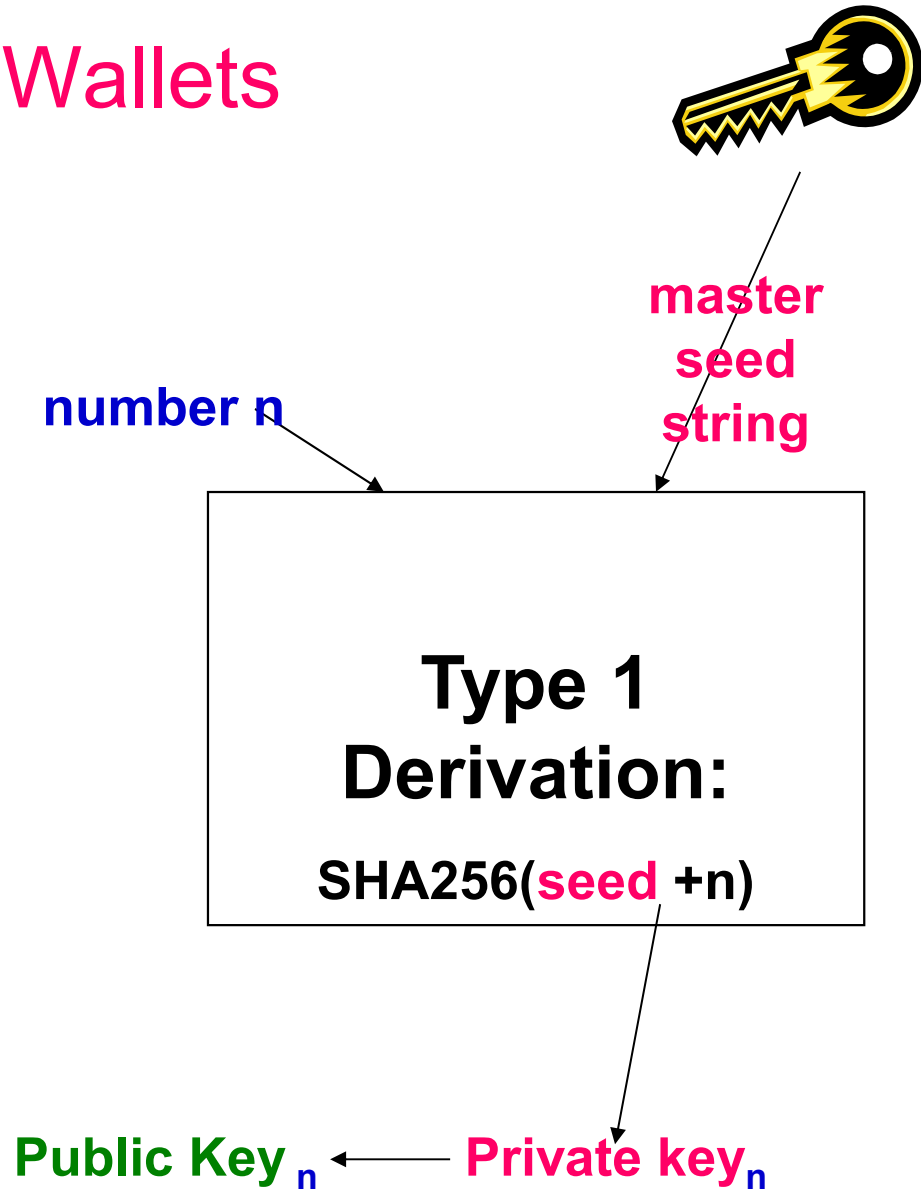
Type 1 Wallets

Proposed by Greg Maxwell.

Simple:

- does not allow “master public keys”
- there is only one chain of keys.

All private keys come from one single secret **seed**.



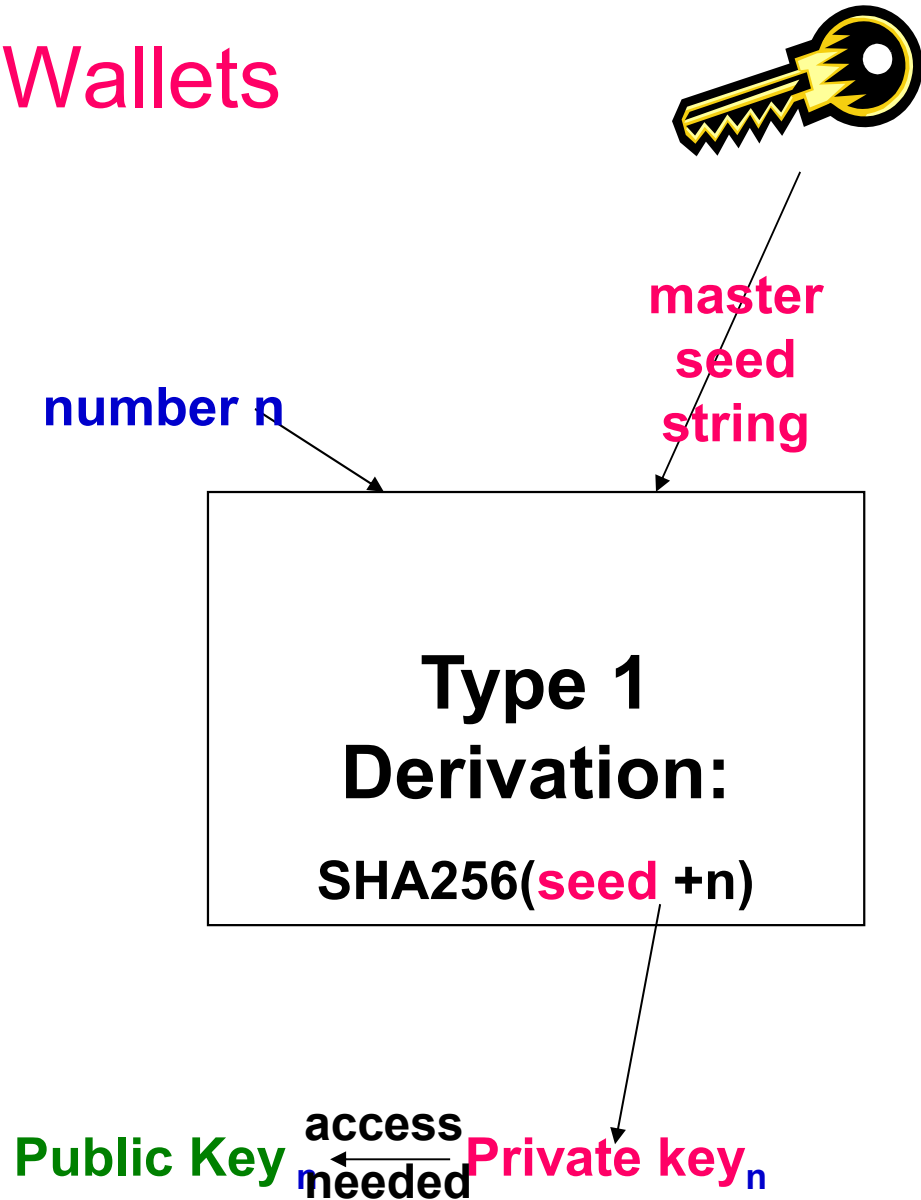
Type 1 Wallets

Proposed by Greg Maxwell.

Simple:

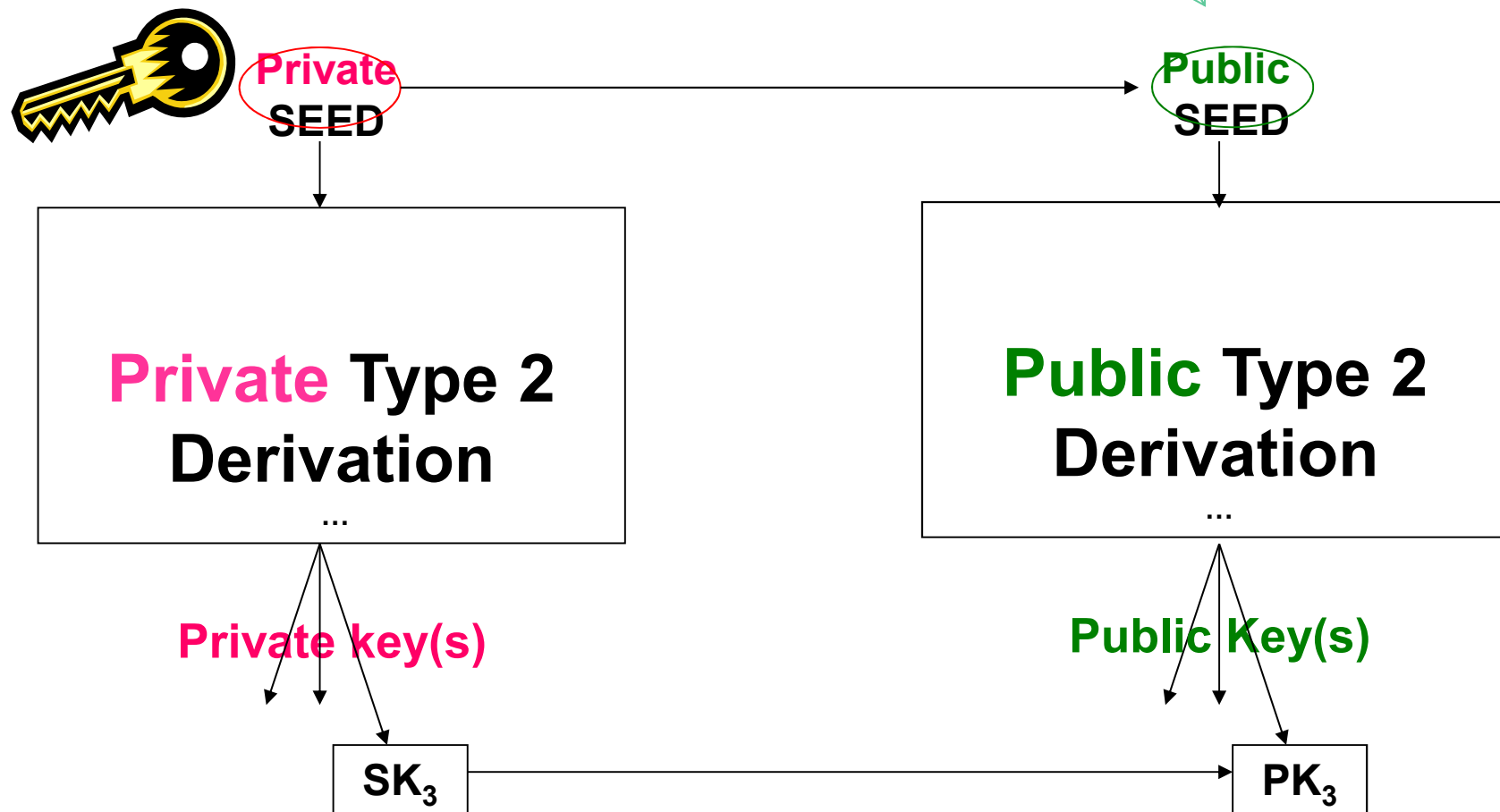
- does not allow “master public keys”
- there is only one chain of keys.

All private keys come from one single secret **seed**.



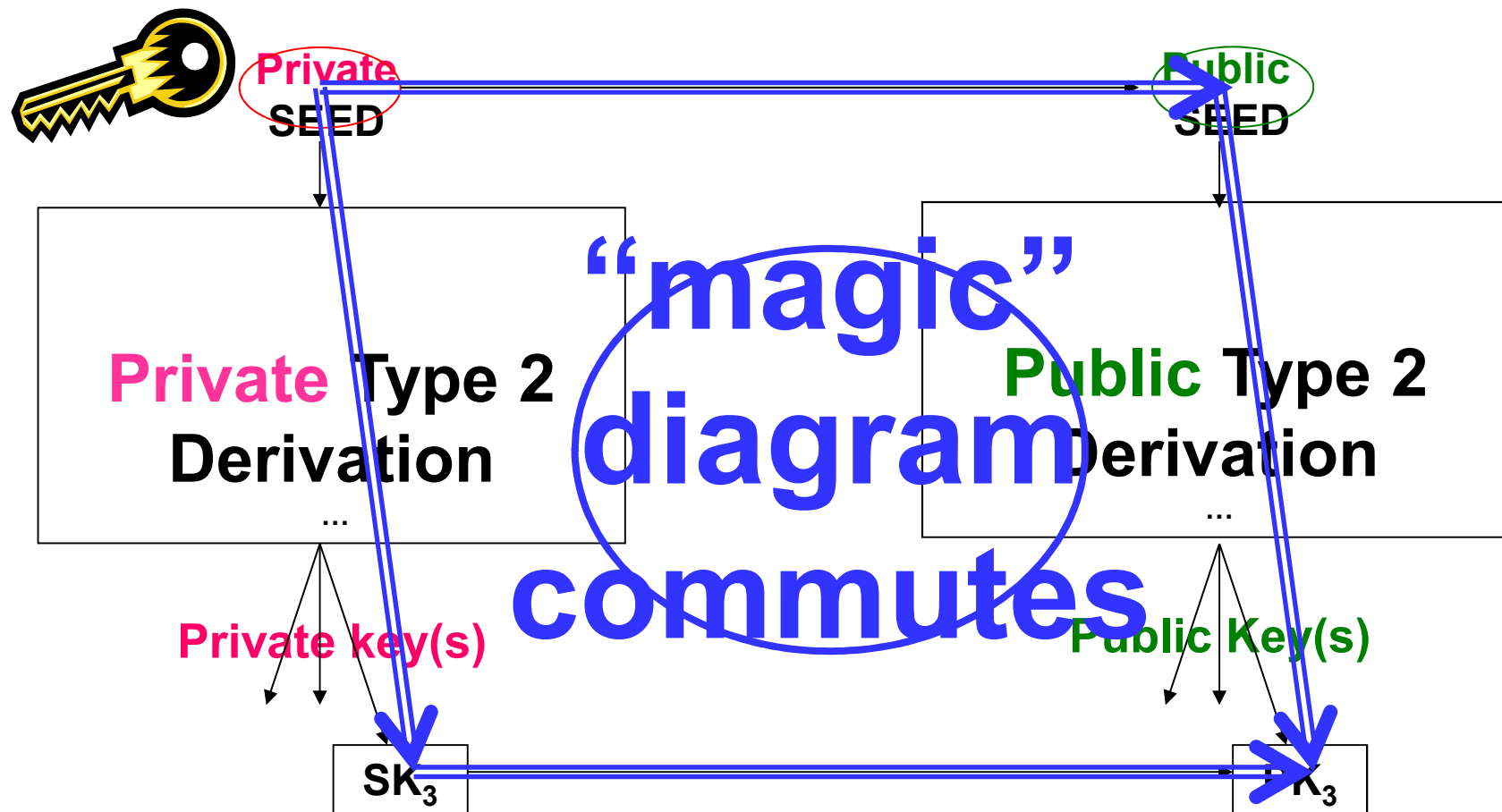
Idea of Type 2 Wallets

[Greg Maxwell, Pieter Wuille] - Has a sort of “master public key”



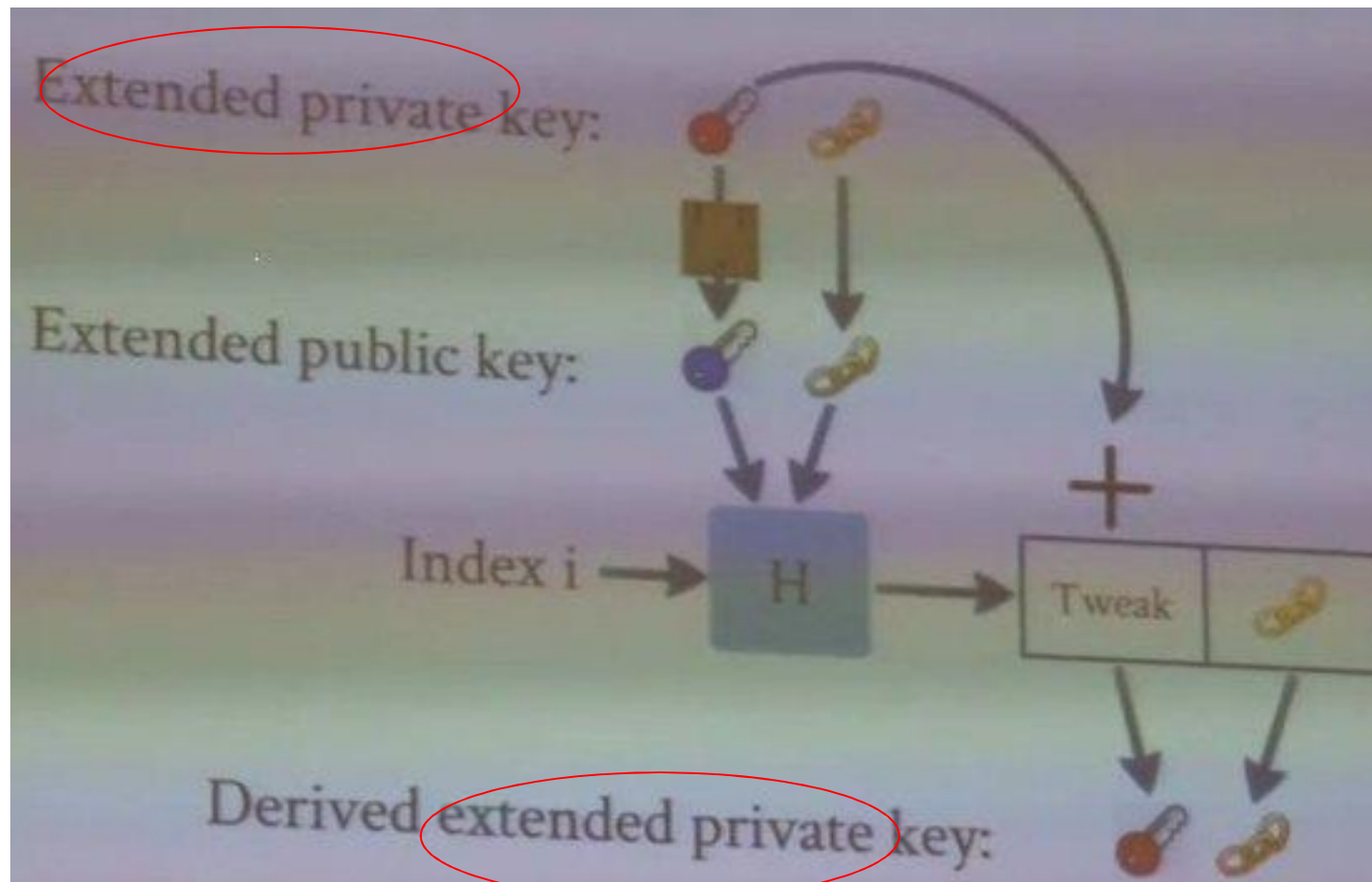
Key “Magical” Property =

both paths give the same keys.



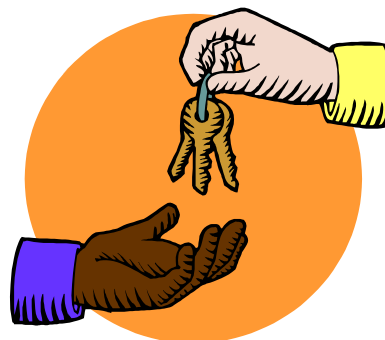
ONE METHOD to Achieve This Magic [Maxwell-Wuille]

PRIV,CHAIN
256+256



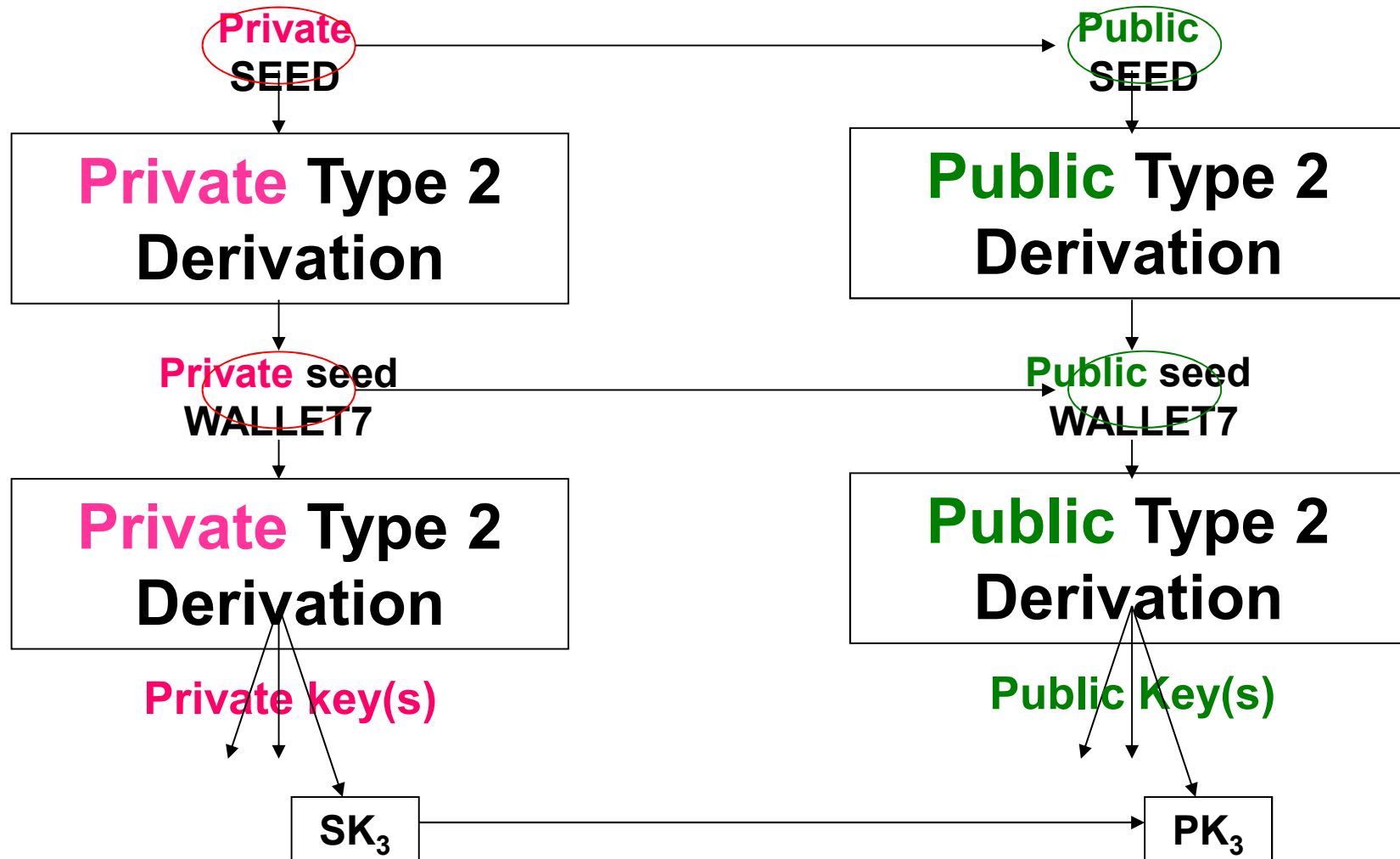
used in
BIP032

Magic on Several Levels



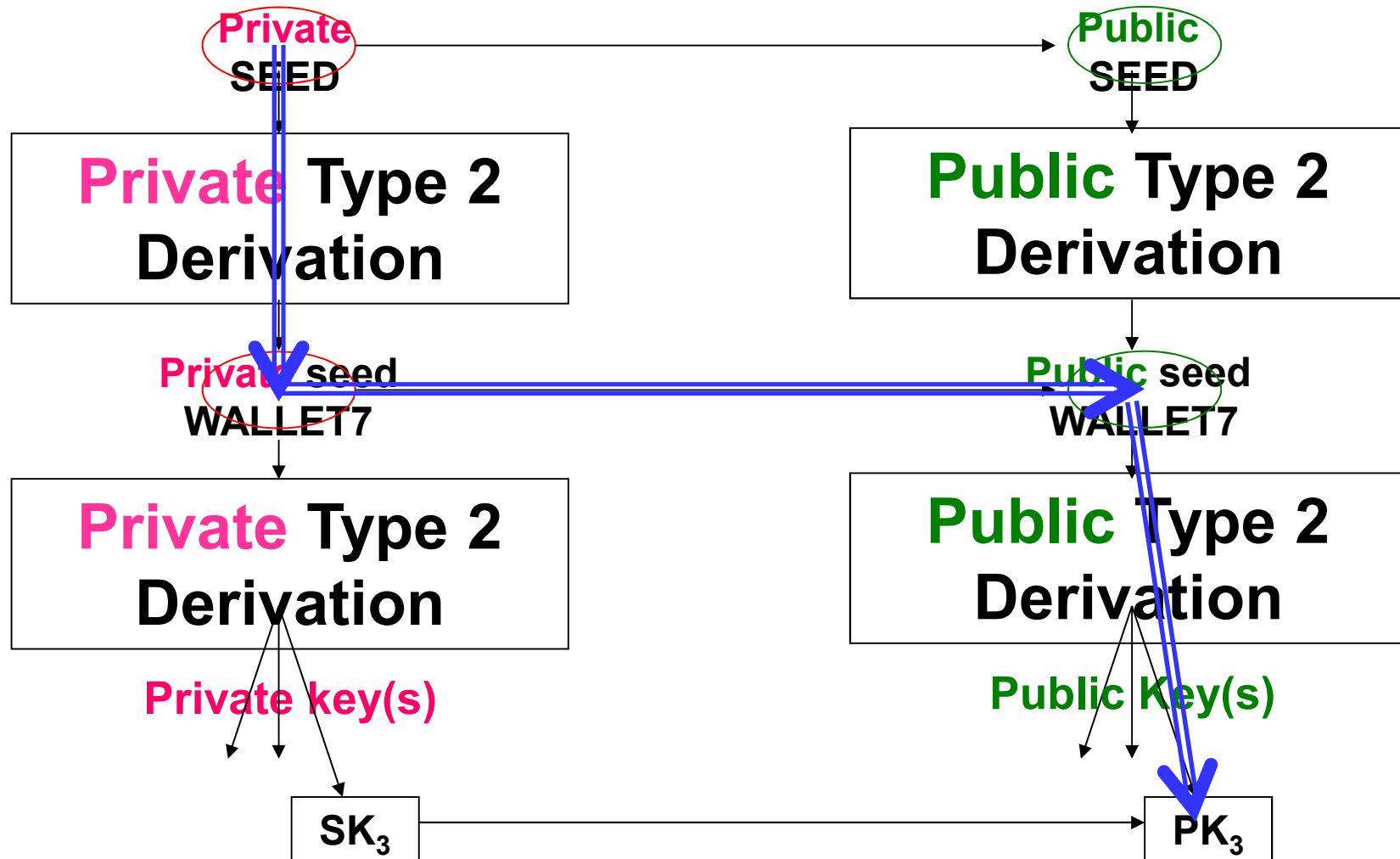
Better: Several Levels! = Hierarchical

=> the magic works over several levels, here 3 levels:



Magic = the Diagram Commutes:

all paths give the same public keys



Def: Sub-domains

Definition: We call **a security sub-domain** a set of all the nodes in this DIRECTED graph above, on any side right/left, with all the nodes and leaves BELOW one point/edge, in the sense of the arrows, which corresponds to what can be computed: lower level keys can be computed from higher level, and public seeds CAN be computed from private seeds, therefore if it is on the left it must also contain some components to the right, but not vice-versa.

For example a domain under some index **i** on ANY side for example **m/1** (left) or **M/0/1=N(m/0/1)** (right).

Benefits



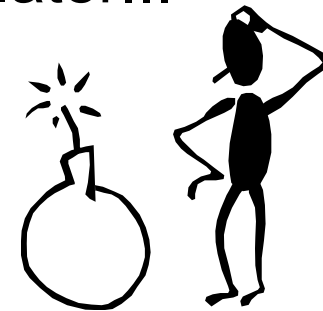
Audit Capability!

One of the main goals and benefits for Type 2.

The auditor can have ALL public keys
and yet no access to funds.



The price to pay is HIGH: more about it later...



E-Commerce Scenario

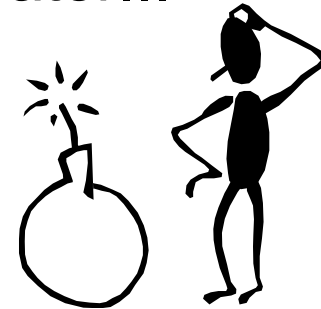
Major frequently cited application is e-commerce.

The receiver of moneys (say e-commerce website) has access to many public keys only AND cannot spend any coins.



AGAIN:

The price to pay is HIGH: more about it later...



Anonymity!

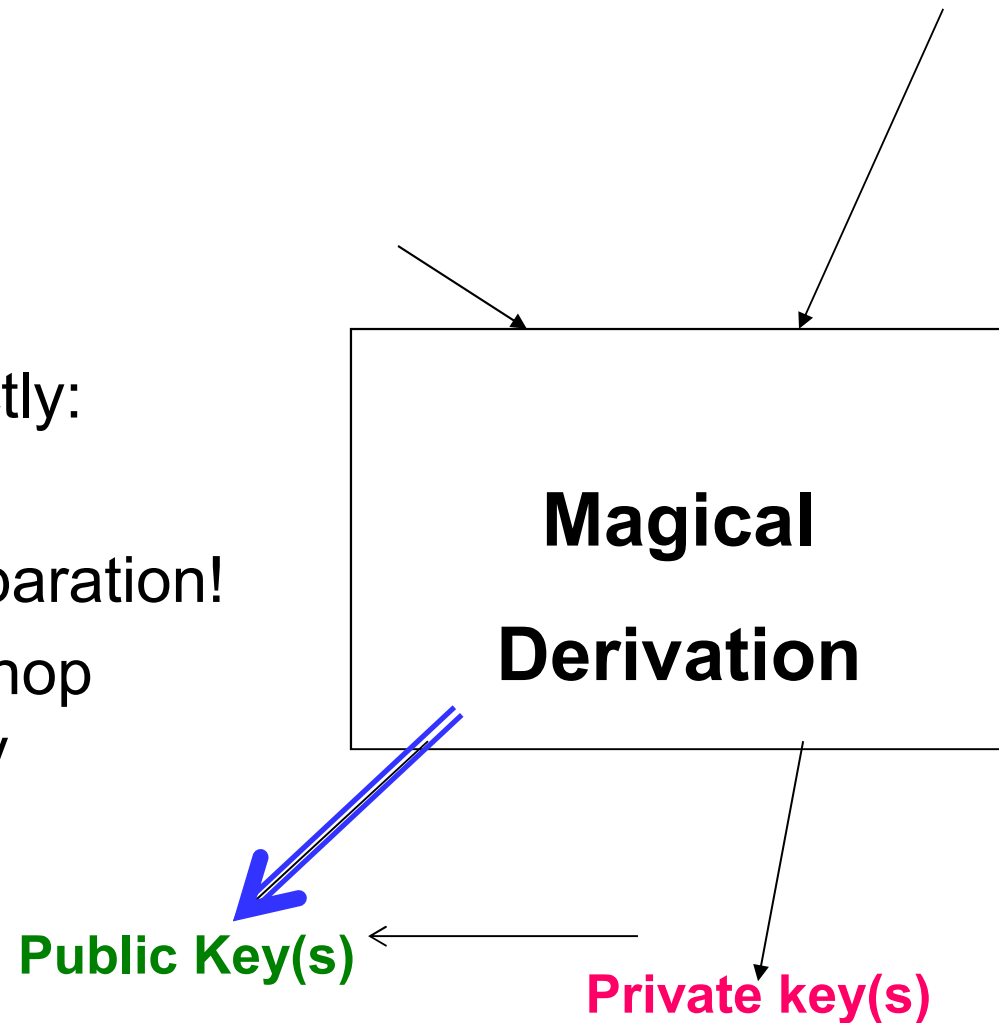
Compute public keys directly:

Very good for privilege separation!

Better **anonymity**: a web shop
can use fresh public key
for each payment

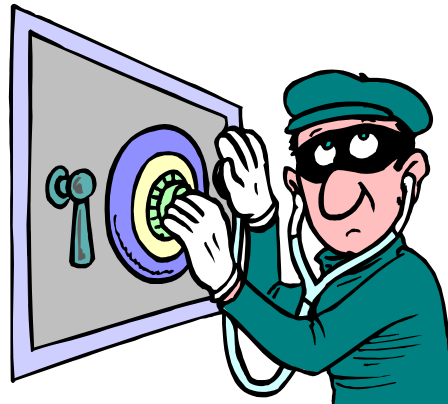
⇒ yet cannot steal the money

⇒ again price to pay is **HIGH**...
cf. security study later...



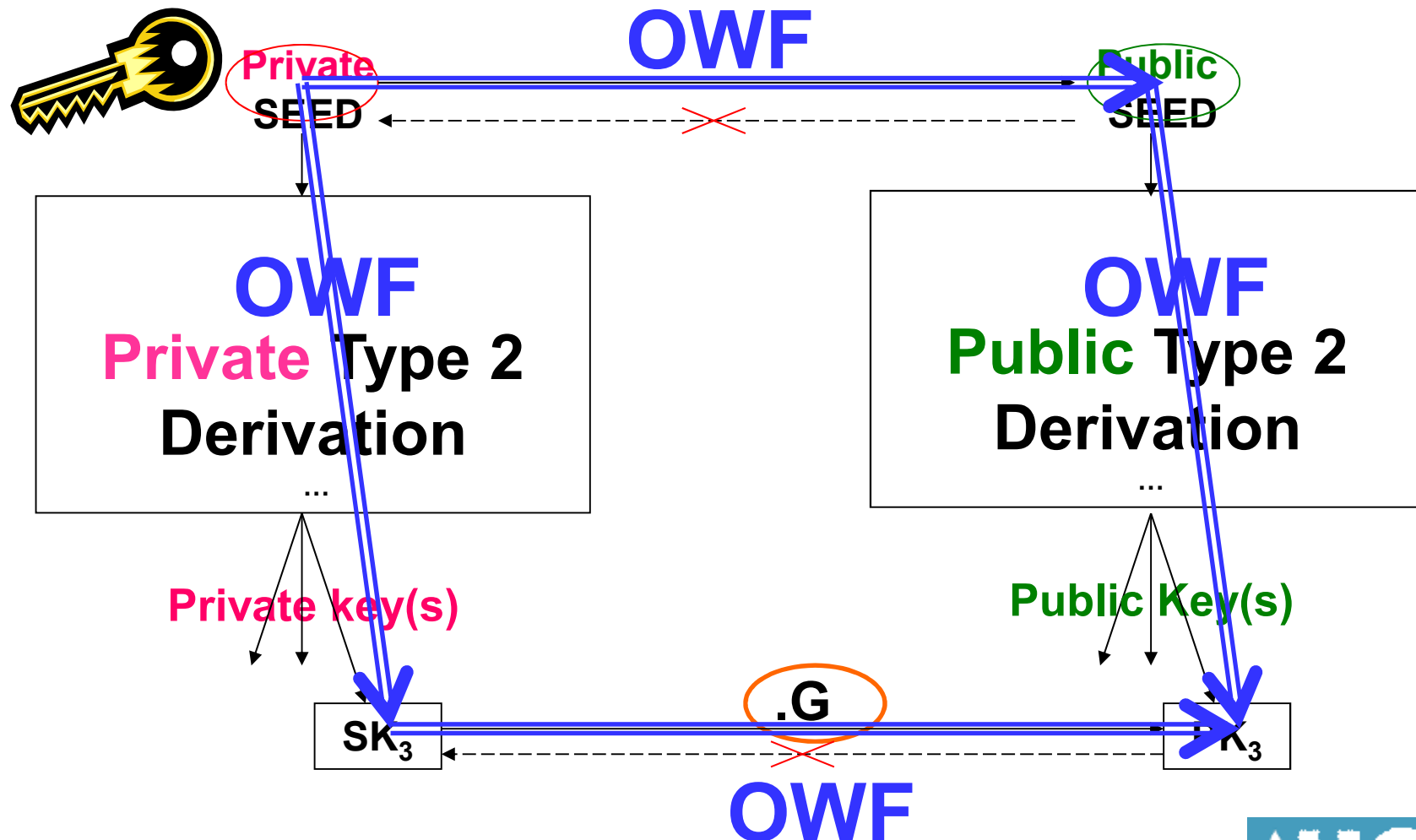
Security of Type 2 HD Wallets

(spec was revised,
most problems remain)

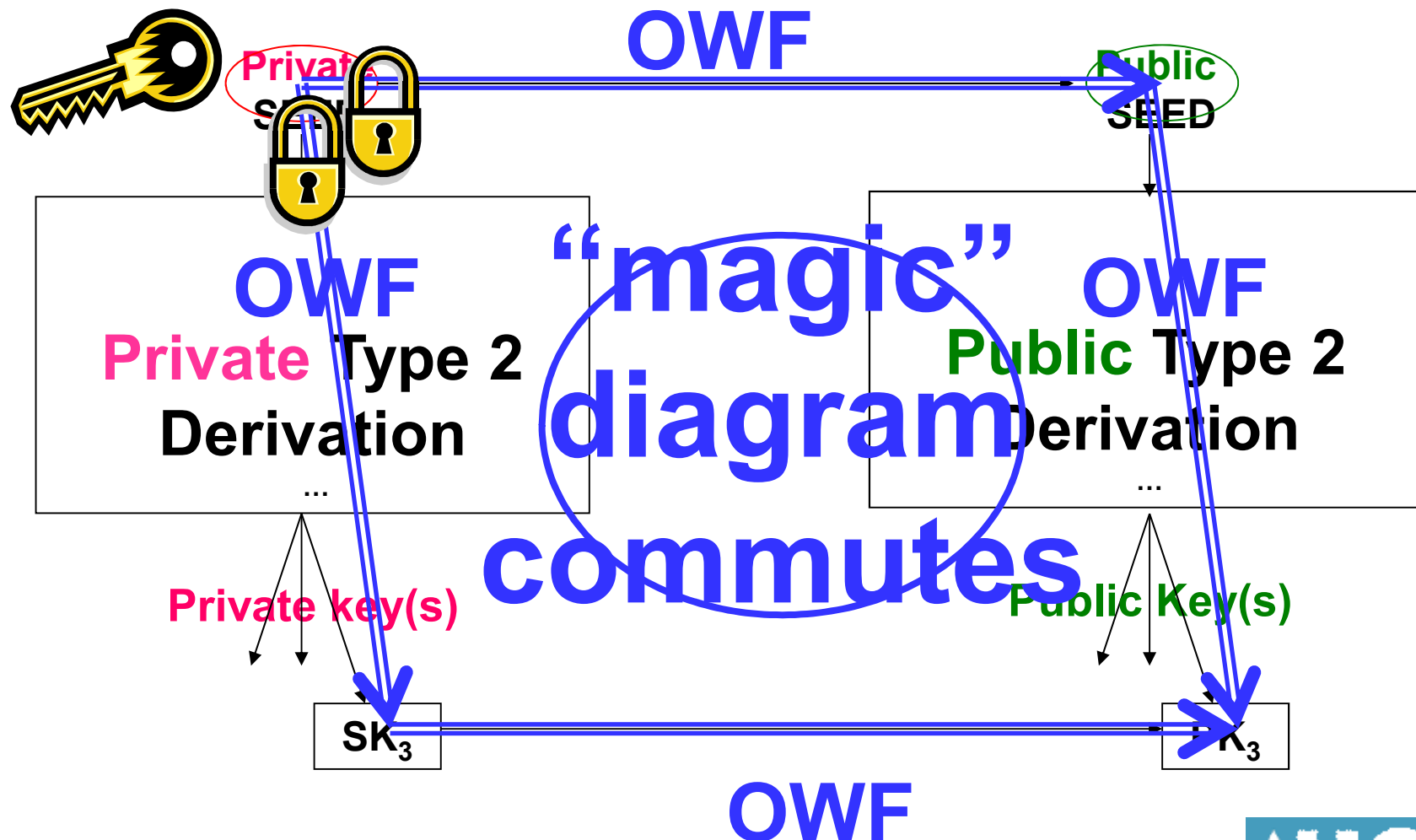


One Way Functions = OWF

no way to go backwards!



Key “Magical” Property =
both paths give the same keys.



Audit Capability => Danger

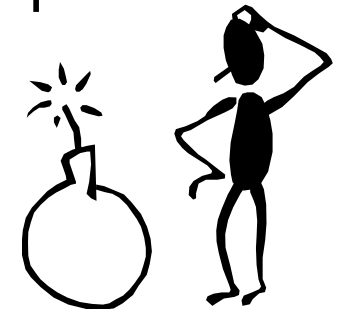
One of the main goals and benefits for Type 2.

The auditor can have ALL public keys
and yet no access to funds.



The price to pay is HIGH:

the auditor **CAN** when collaborating with 1 holder of private key, **recover LOTS of other private keys**
[up to a total compromise of ALL master keys].



There exist alternatives!

=> Give the auditor lots of public keys, no harm possible ever!

E-Commerce Scenario?

Major frequently cited application is e-commerce.

The receiver of moneys (say e-commerce website) has access to many public keys only AND cannot spend any coins.

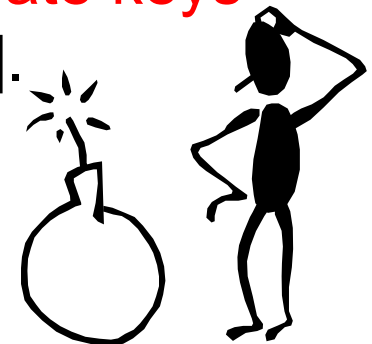


AGAIN:

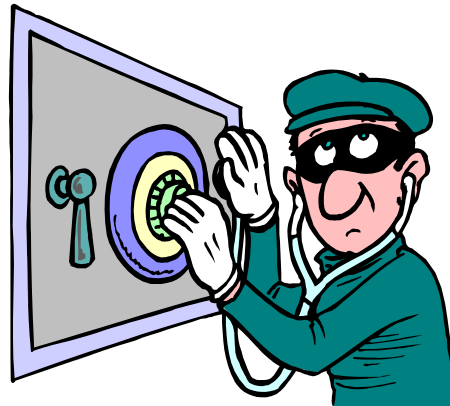
The price to pay is HIGH:

the e-commerce website can be hacked with SQL injection or similar and the attacker **CAN** when collaborating with just 1 holder of private key, **recover LOTS of other private keys** [up to a total compromise of ALL master keys].

Again a safer alternative is to give the e-commerce merchant lots of public keys, no harm possible.

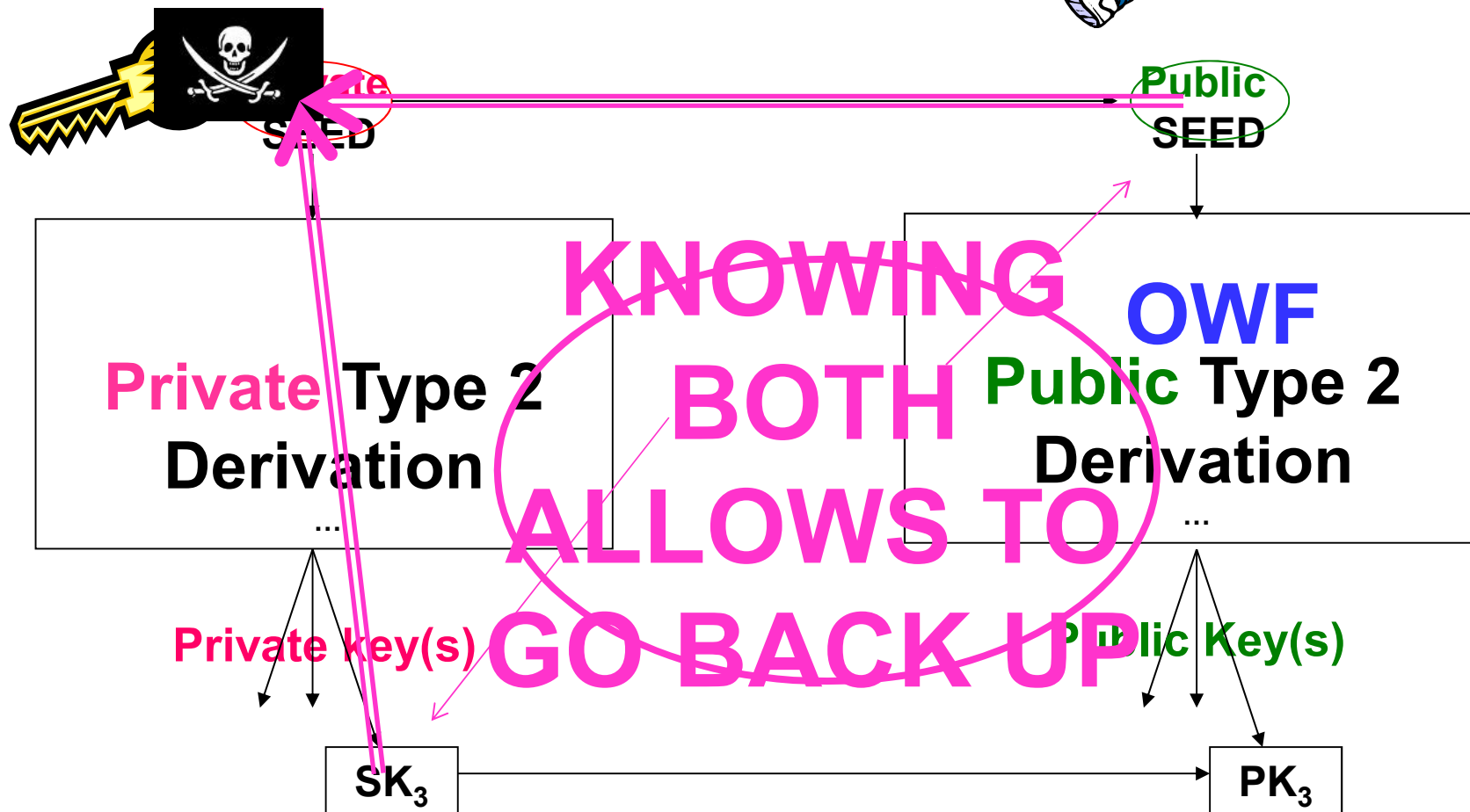


Key weakness

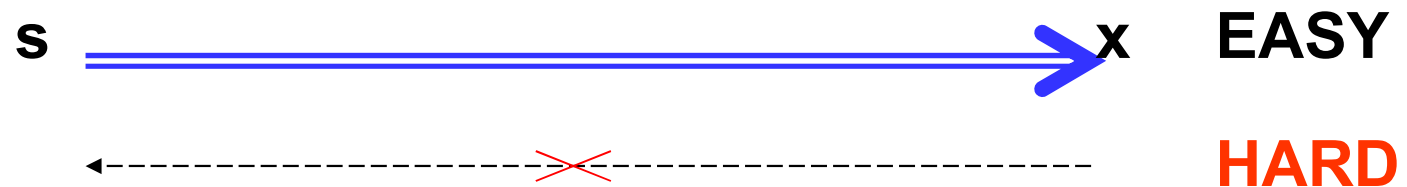


KEY WEAKNESS

two secrets reveal EVERYTHING!

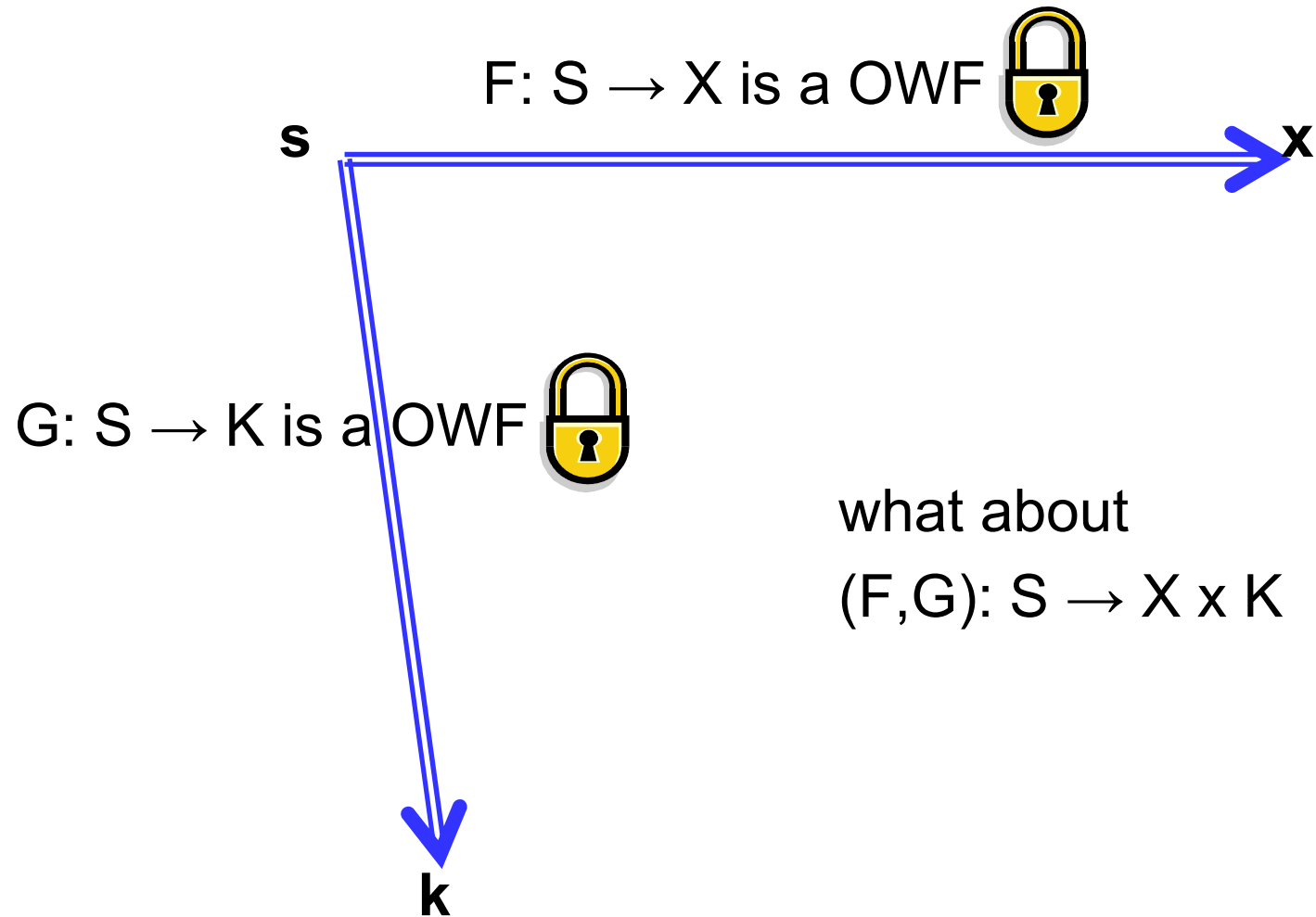


Mathematical Formulation

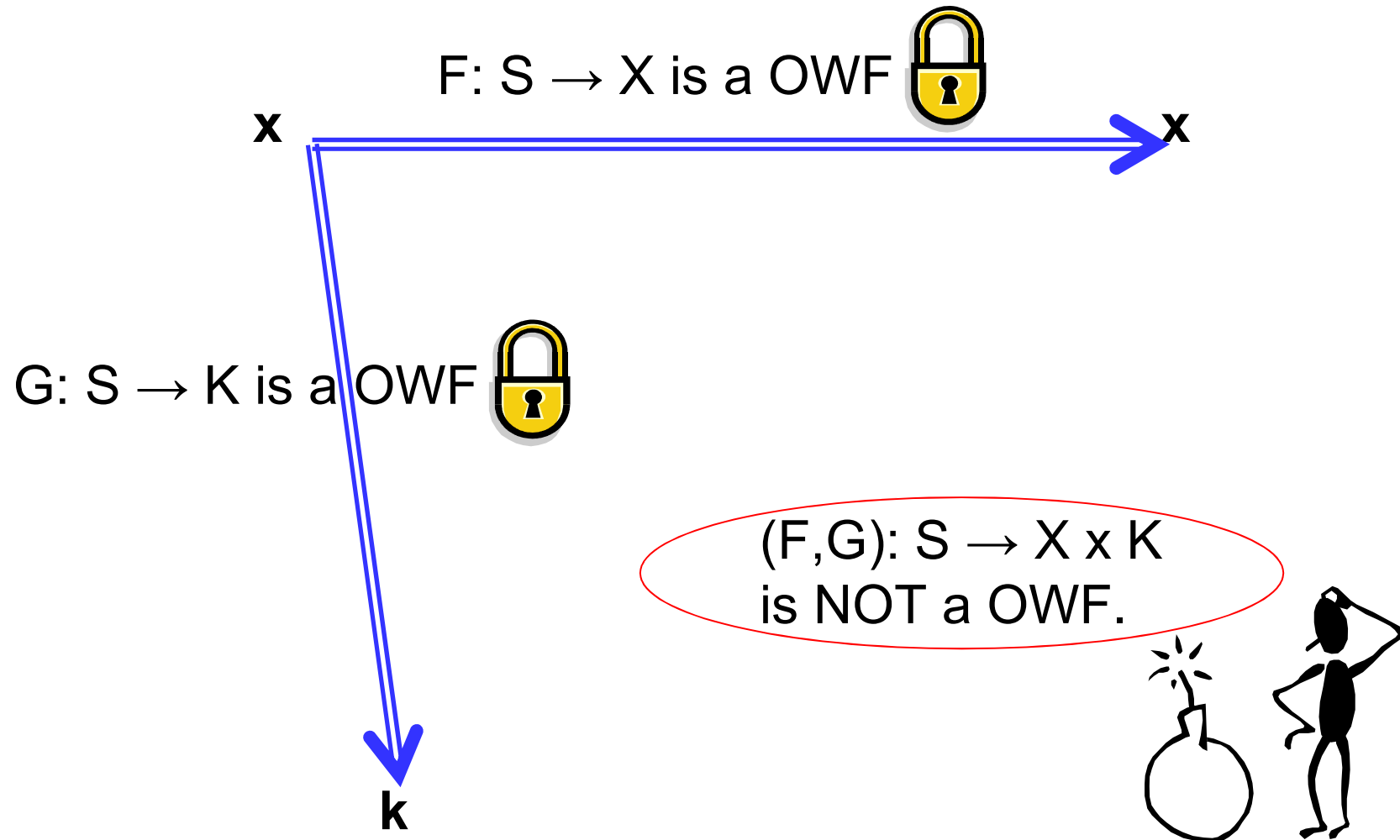


$F: S \rightarrow X$ is a OWF 

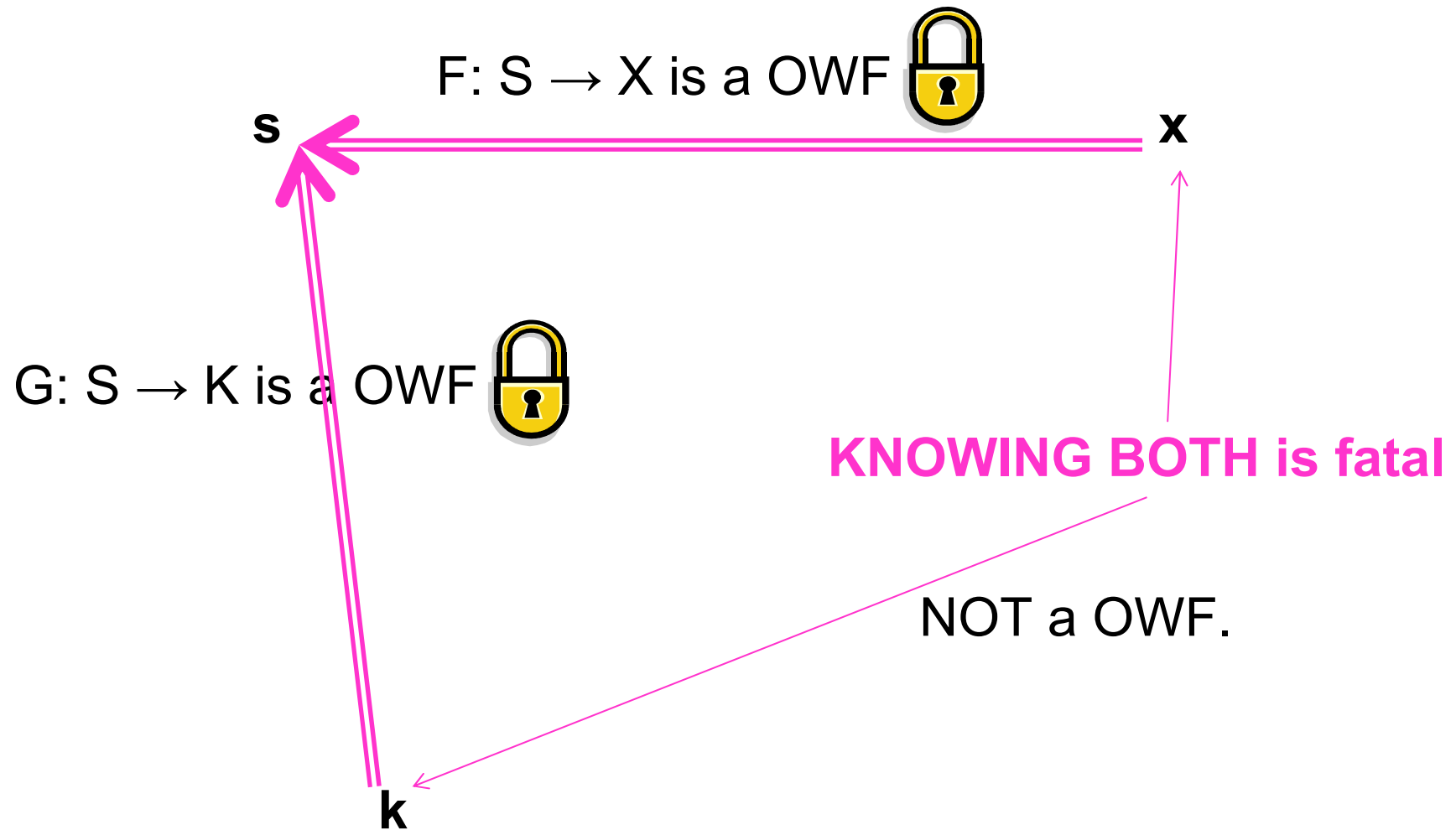
Mathematical Formulation



Mathematical Formulation

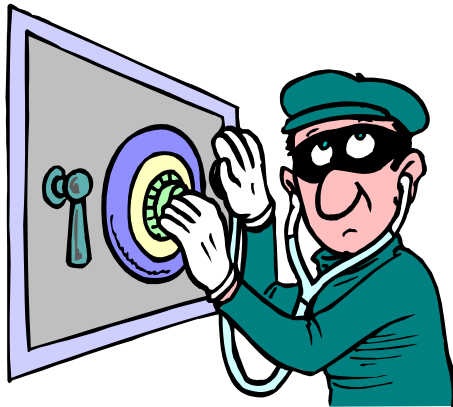


Mathematical Formulation



Criticism of Type 2 HD in Blogs

(spec was revised,
most problems remain)



Citation From BIP032

“elliptic curve mathematics permit schemes where one can calculate the public keys without revealing the private keys”

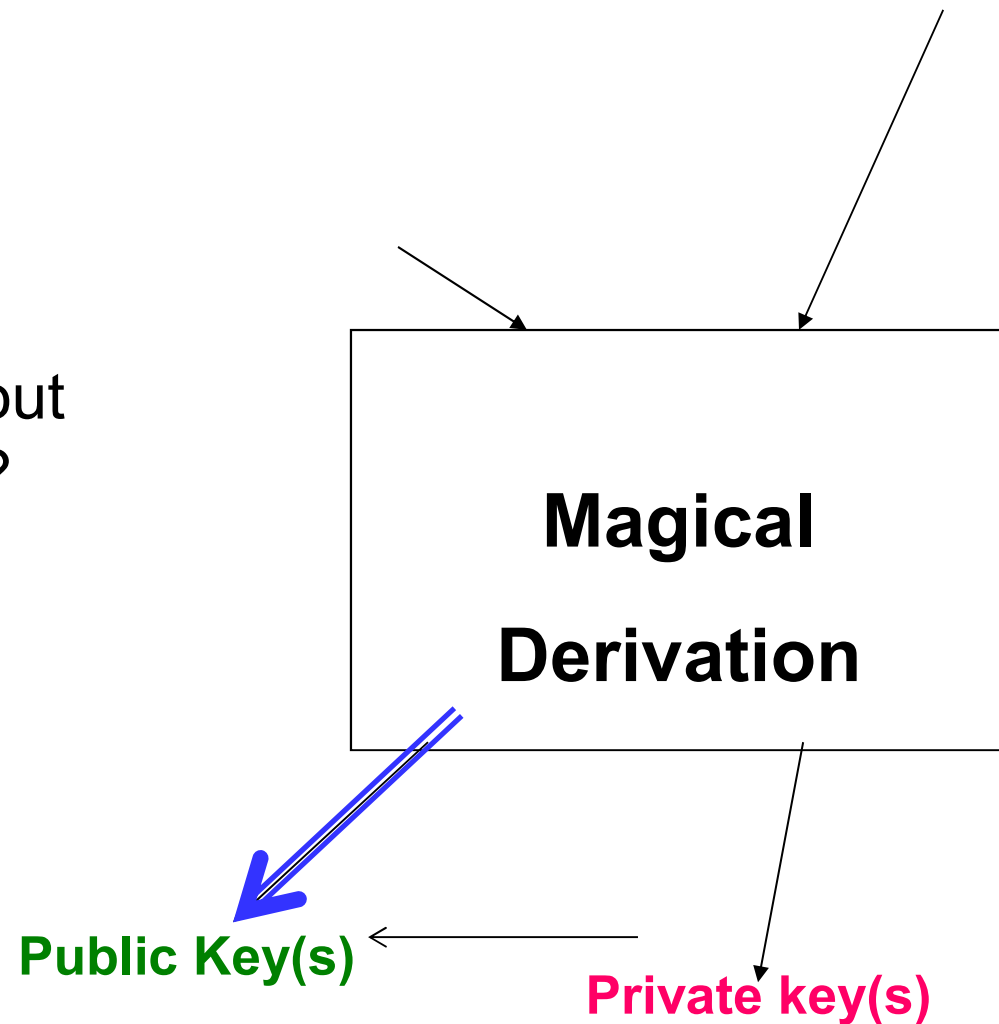
Really????

Yes, but it is not very secure...

ECC Magic

How can
“master public keys”
exist?

Compute public keys without
seeing the private keys?



ECC Magic – Basic Principle

How can
“master public keys”
exist?

**S =
extra
secret**

n

“Key Homomorphism”

“**Magic**”

PrivateKey(type,n) = Master_private_key + H(n|**S**|type) mod q

PublicKey(type,n) = Master_public_key + H(n|**S**|type)***G**

Public Key(s)

Private key(s)

Leaks



Warning

The magic is NOT very good.

There are many compromise scenarios.



Leaks Upwards?

Remark: not quite true in BIP032, audit key also needed...

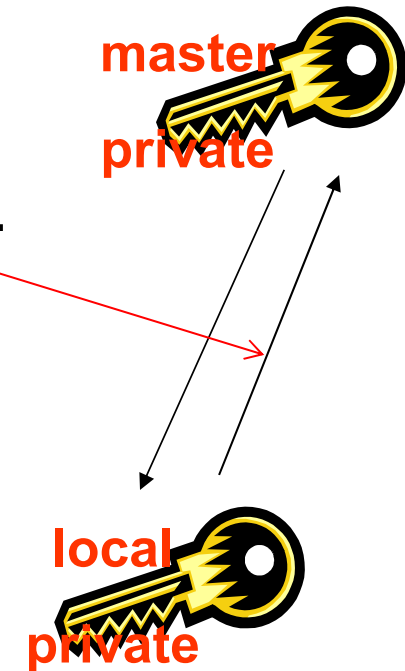
Could one private key leak the MASTER private key???

In BIP032 problem is mitigated by additional secrets...

these are later called **chain codes**

⇒ these are not so well kept secrets

- ⇒ known to auditors for example,
- ⇒ many of them will be known to many hot (exposed) web servers ...
- ⇒ etc.



*Remark

Slides with * can be omitted

*Serious Problem For Simplified Spec

Huge dangers with sharing private keys, ever,

⇒ for example when funding your Mt Gox account with the "Redeem private key" option, it gives the key to Gox!!

⇒ Pb: **one private key compromises all, potentially!!!**

(again one audit key also needed... not hard to get)

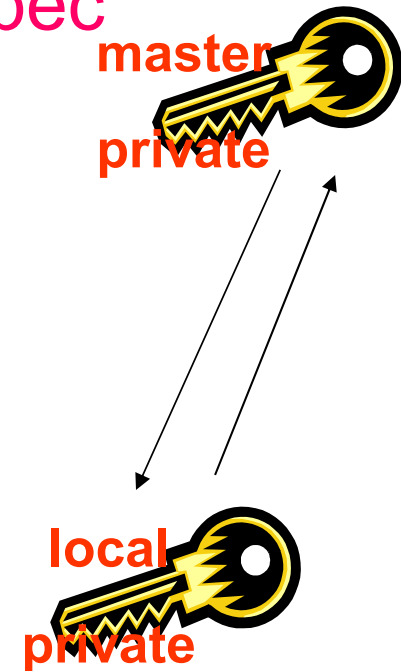
user Danke in forum: Re: Deterministic wallets
October 02, 2012, 03:37:05 AM

The use scenario is that a "hot" machine (connected to the Internet and potentially vulnerable) generates a chain of public keys, and for that purpose stores **Master_public_key** as well as **S**.

Explanation: Here **S**=extra secret, in BIP032 it will be called a **chain code** and denoted by **C**

In a reasonable threat model **both values** become available to an attacker because the "hot" machine is vulnerable.

(continued next slide)



*Serious Problem For Simplified Spec

[...] stores **Master_public_key** as well as **S**[...]

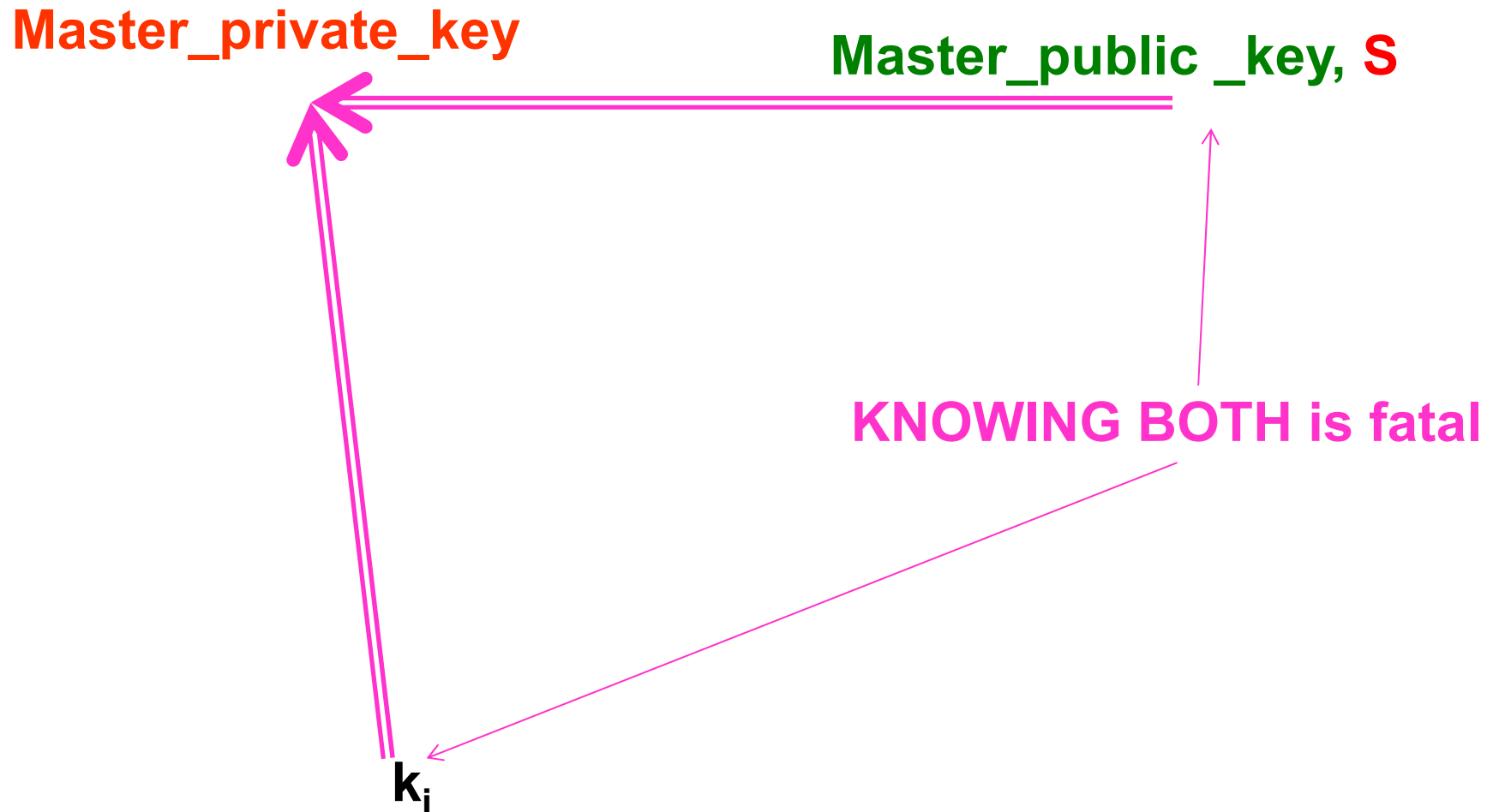
[...] **both values** become available[...]

Since [...] the serial numbers **n** and the **type** can be guessed, a single leaked private key will leak **Master_private_key** as well by a simple subtraction in the finite field underlying secp256k1.



$$\begin{aligned} \text{PrivateKey}(\text{type}, n) &= \text{Master_private_key} + H(n|\mathbf{S}|\text{type}) \bmod q \\ \text{PublicKey}(\text{type}, n) &= \text{Master_public_key} + H(n|\mathbf{S}|\text{type}) * \mathbf{G} \end{aligned}$$

*BTW: It's the same attack as before



*Serious Problem For Simplified Spec

both Master_public_key and S known

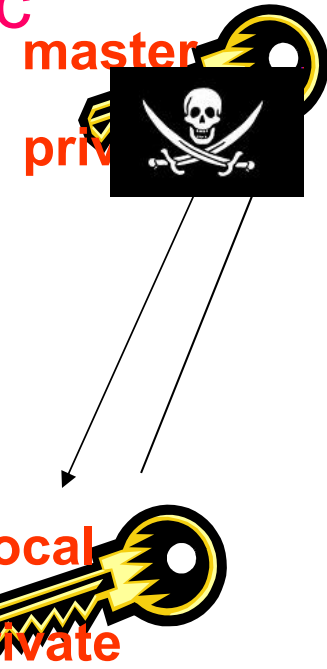
=>

a single leaked private key will leak Master_private_key

Conclusion: Users should be advised that ANY private key from the generated key chain is to be secured with the same security level as the master private key.

This seems to affect not only the implementation of deterministic wallets in electrum and armory but also the hierarchical deterministic wallets suggested in BIP 032s

[how? continued next page]:



***Serious Problem For Actual BIP032!**

THE SAME ATTACK still works.

*Serious Problem For Old/Actual BIP032!

(quoting the spec of BIP32 from elsewhere):

We define $\text{CKD}((k_{\text{par}}, c_{\text{par}}), n)$ [...]

takes a parent extended **secret** key $(k_{\text{par}}, c_{\text{par}})$ and a 32-bit integer n ,
to produce a child extended secret key (k_n, c_n) .

Call $I = \text{HMAC-SHA512}(\text{Key}=c_{\text{par}}, \text{Data}=k_{\text{par}}*G \parallel n)$, (512 bits)

where $k_{\text{par}}*G$ is the public key corresponding to k_{par} ,

\parallel is concatenation, n is 32 bits unsigned, most sign. byte first.

Split I into two 32-byte sequences, IL and IR .

- k_n is equal to $IL * k_{\text{par}}$.
- c_n is equal to IR .

out of date, changed to+ 30 Apr 2013
but the attack works the same...

There is also a version that operates on extended **public** keys instead of private ones:

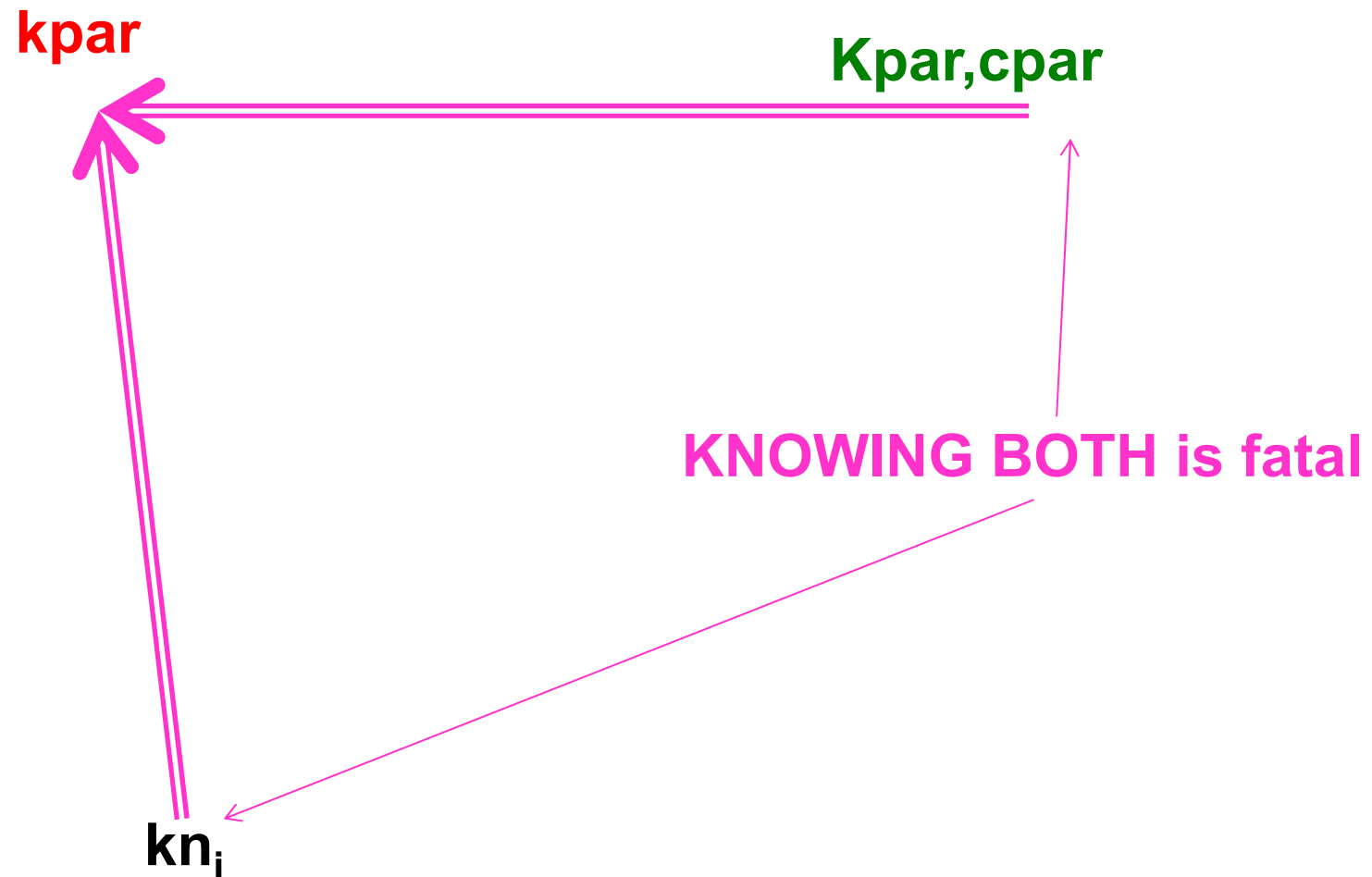
$(K_n, c_n) = \text{CKD}'((K_{\text{par}}, c_{\text{par}}), n)$:

Call $I = \text{HMAC-SHA512}(\text{Key}=c_{\text{par}}, \text{Data}=K_{\text{par}} \parallel n)$, (512 bits)

Split I into two 256-bit 32-byte sequences, IL and IR .

- K_n is equal to $IL * K_{\text{par}}$.
- c_n is equal to IR .

*again the same attack!



*So What?

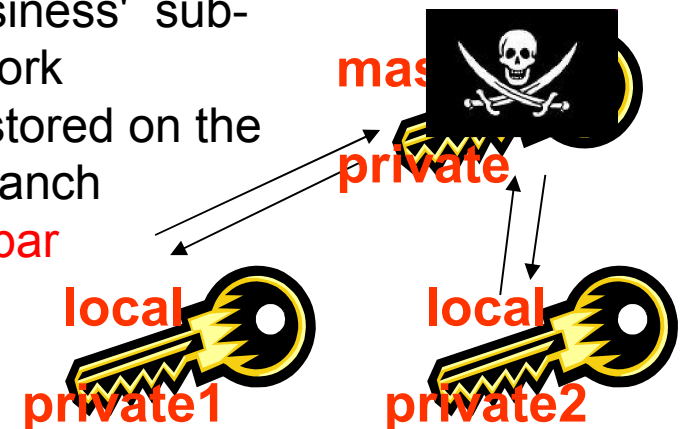
[...] "hot" machine of some main business wants to be able to automatically generate extended public keys (K_n, c_n) . It needs to store (K_{par}, c_{par}) to do that.

Suppose it computes the corresponding extended secret key (k_n, c_n) on a "cold" machine and hands it to the business' sub-branch numbered n , so that the sub-branch can work independently. Now, if the information (K_{par}, c_{par}) stored on the main business' "hot" machine leaks then the sub-branch can obtain the main business master private key k_{par} by these computations:

$I = \text{HMAC-SHA512}(\text{Key} = c_{par}, \text{Data} = K_{par} || n)$

Split I into two 32-byte sequences, IL and IR

$k_{par} = IL^{-1} * k_n$ (arithmetic in the finite field underlying secp256k1)



**PARENT PRIVATE KEY IS
REVEALED!**

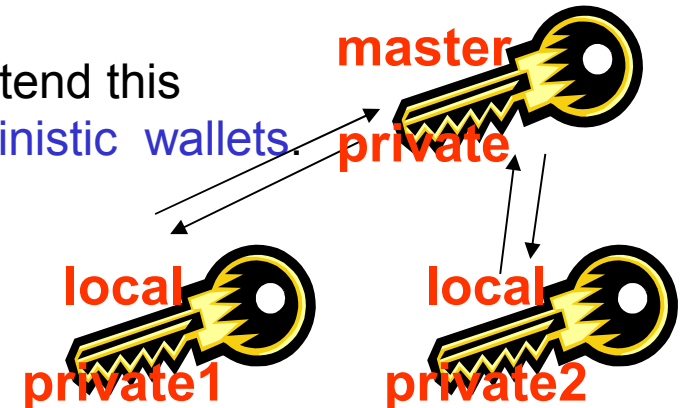


*So What?

Conclusion:

derived secret keys are handed over to sub-branches
=> the derived public keys **MUST** not be generated
on the fly on "hot" machines.

Instead, even **K_{par}** must be stored "cold". To some extent this
contradicts the original idea of hierarchical deterministic wallets.



Garzik

Jeff Garzik wrote [Dec 2012]:

In general, deterministic wallets (a) make life easier while
(b) *increasing* the security threat (e.g. **decreasing** overall security)

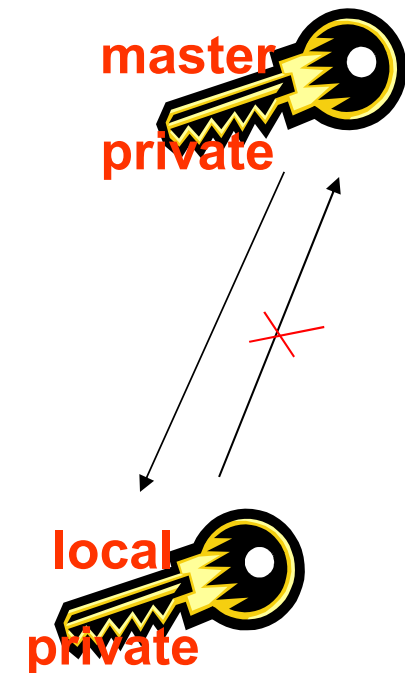
<https://bitcointalk.org/index.php?topic=19137.msg1380099#msg1380099>

BIP032 Revisions

April 2013:

spec was revised:

- * replaced by+
 - Obligatory upgrade, compatibility lost
- “hardened” derivation functions added
 - optional, backwards compatible



Hardened Versions

hardened version

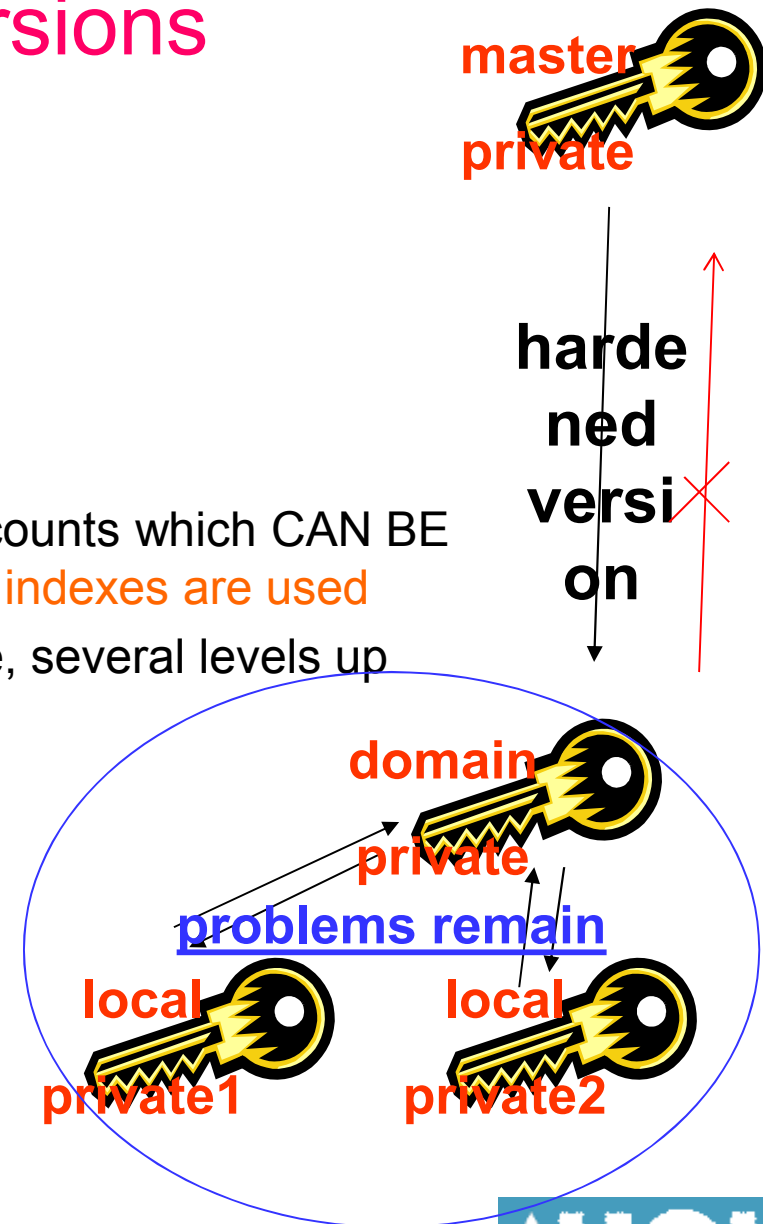
locally removes the “magic on public keys”

More Secure?

Yes and no,

- they have several distinct domains/wallets/accounts which CAN BE well insulated, **only at places where hardened indexes are used**
- but inside these domains there will be leakage, several levels up

More about security of hardened versions later...



Design and Evolution of BIP032 Type 2 HD Wallets

Type 2 HD Wallets

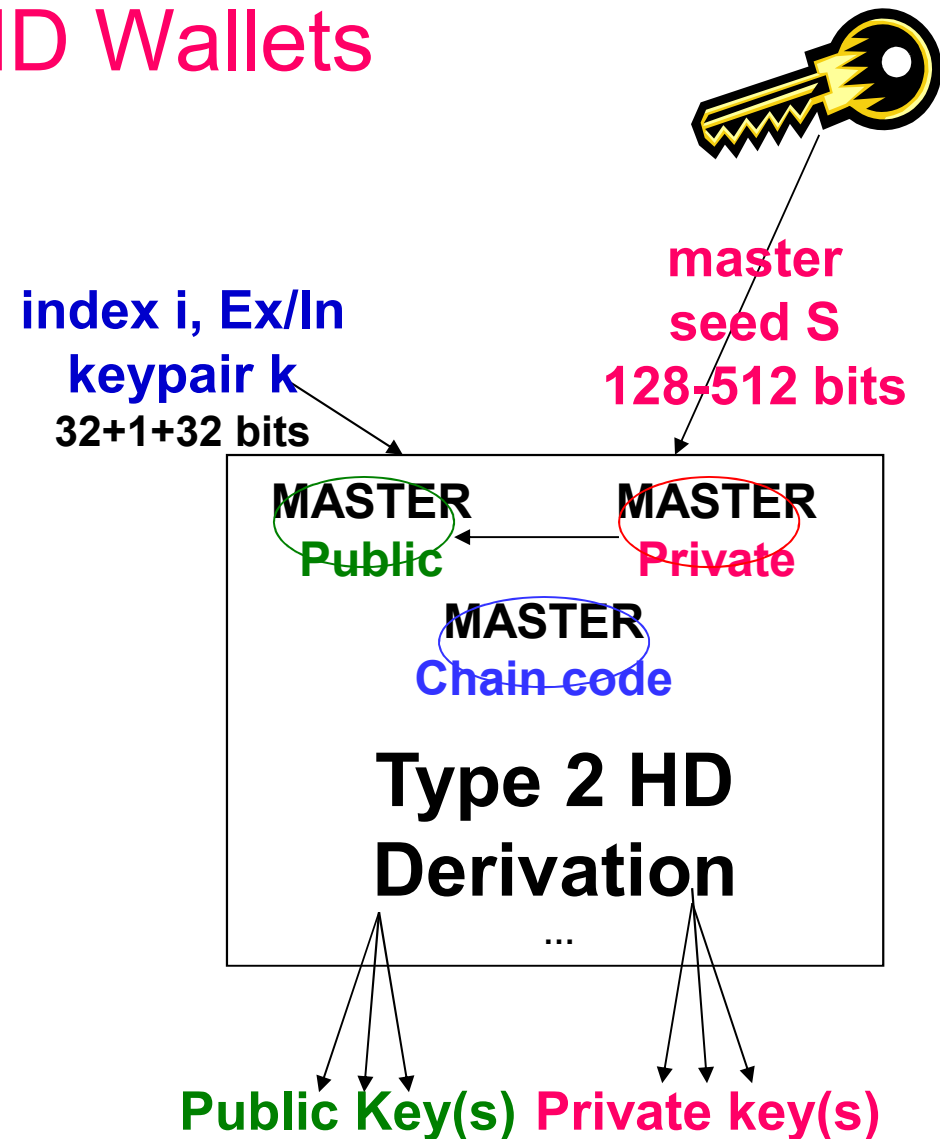
Has

“master public keys”

Current solution:

BIP0032 [Pieter Wuille]

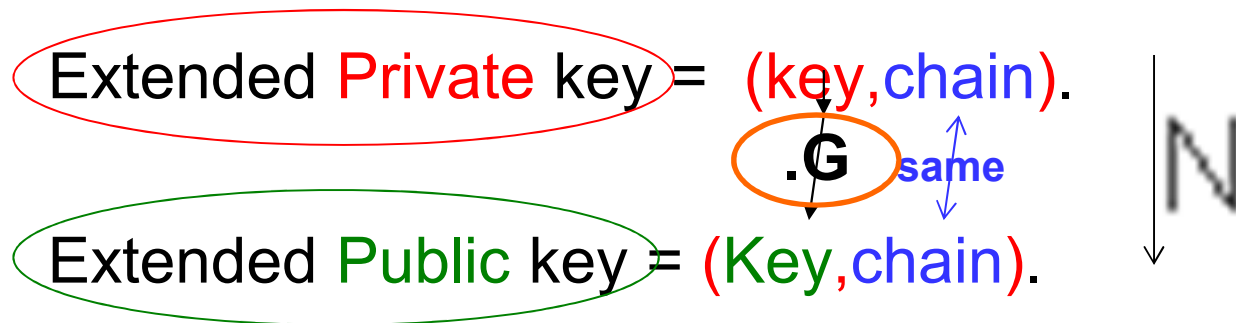
- multiple 4 or more levels
- extended secrets:
keys + chain codes **c**
- first we compare older and new versions of it...



Def. Extended Keys

Two 256-bit secrets.

Knowing just one is secret,
is not enough to reveal it.

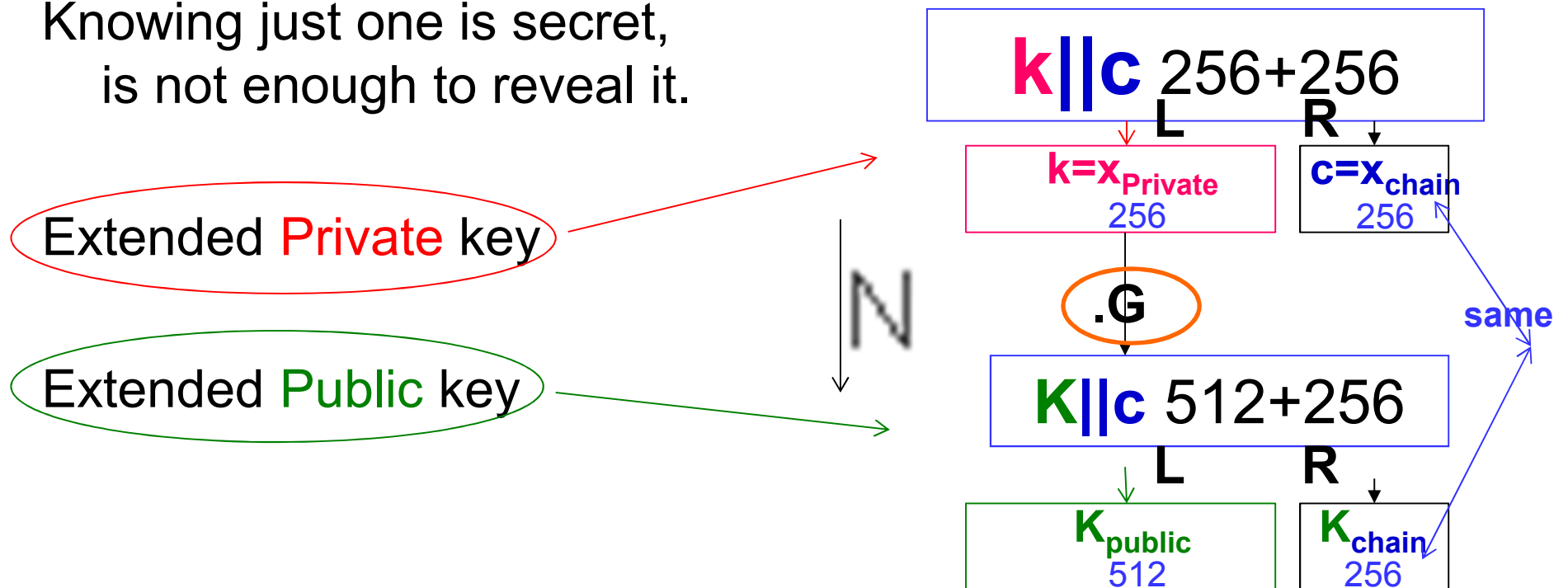


Neutered version,
cannot spend

Def. Extended Keys

Two 256-bit secrets.

Knowing just one is secret,
is not enough to reveal it.



Extended Keys - Example

Uses Pycoin lib by Richard Kiss

>ku P:foo => produces two 111-digit base58 strings!!!!

wallet key:

xprv9s21ZrQH143K31AgNK5pyVvW23gHnkBq2wh5aEk6g1s496M8ZMjxncCKZKgb5j\
ZoY5eSJMj2Vbyvi2hbmQnCuHBujZ2WXGTux1X2k9Krdtq

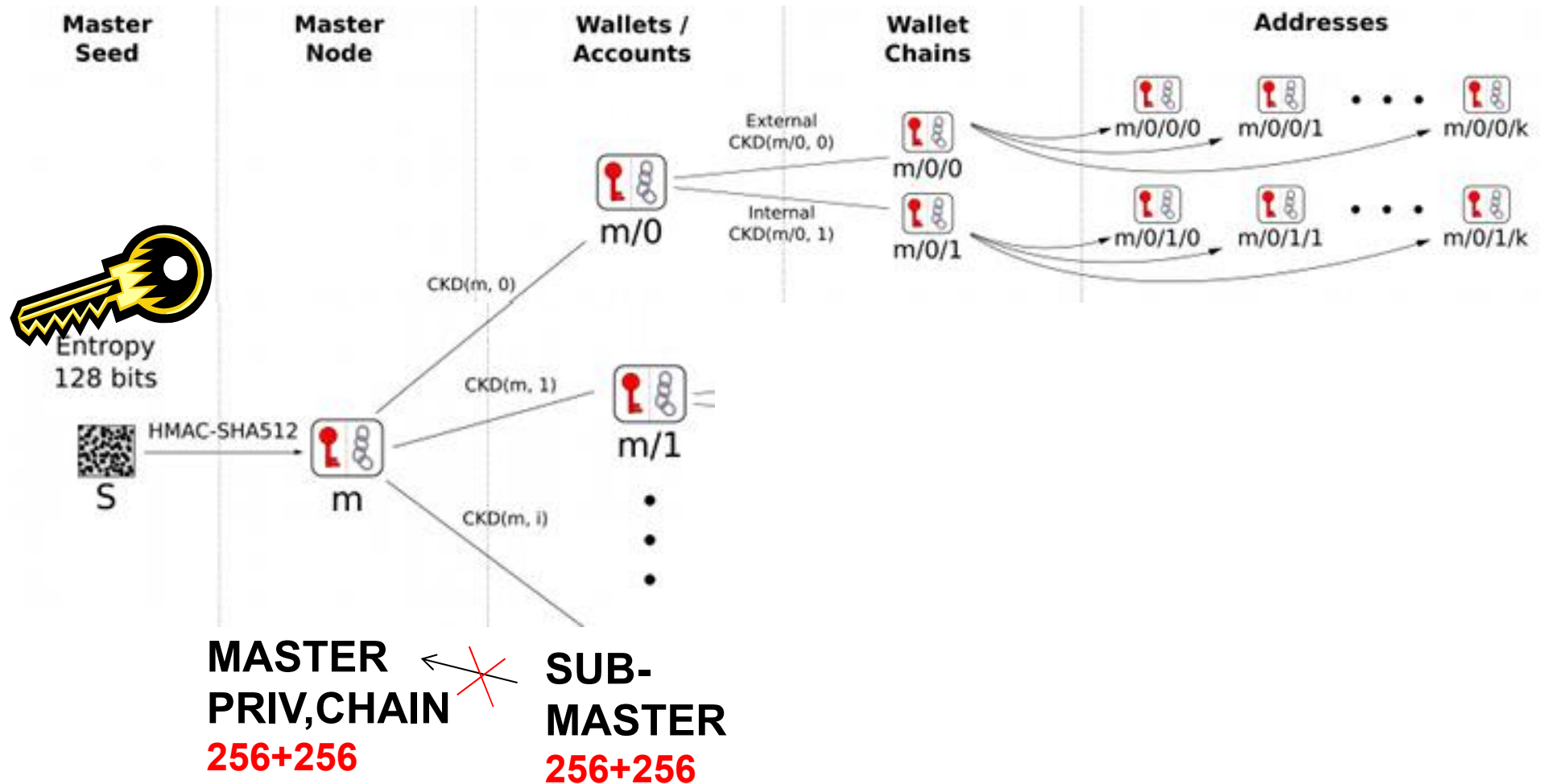
public version:

xpub661MyMwAqRbcFVF9ULcqLdsEa5WnCCugQAcgNd9iEMQ31tgH6u4DLQWoQayvtS\
VYFvXz2vPPpbXE1qpjoUFidhjFj82pVShWu9curWmb2zy

↓
N

This example is continued in LATER slides...

BIP032 – 4 Levels



Generic Solutions

There are TWO!

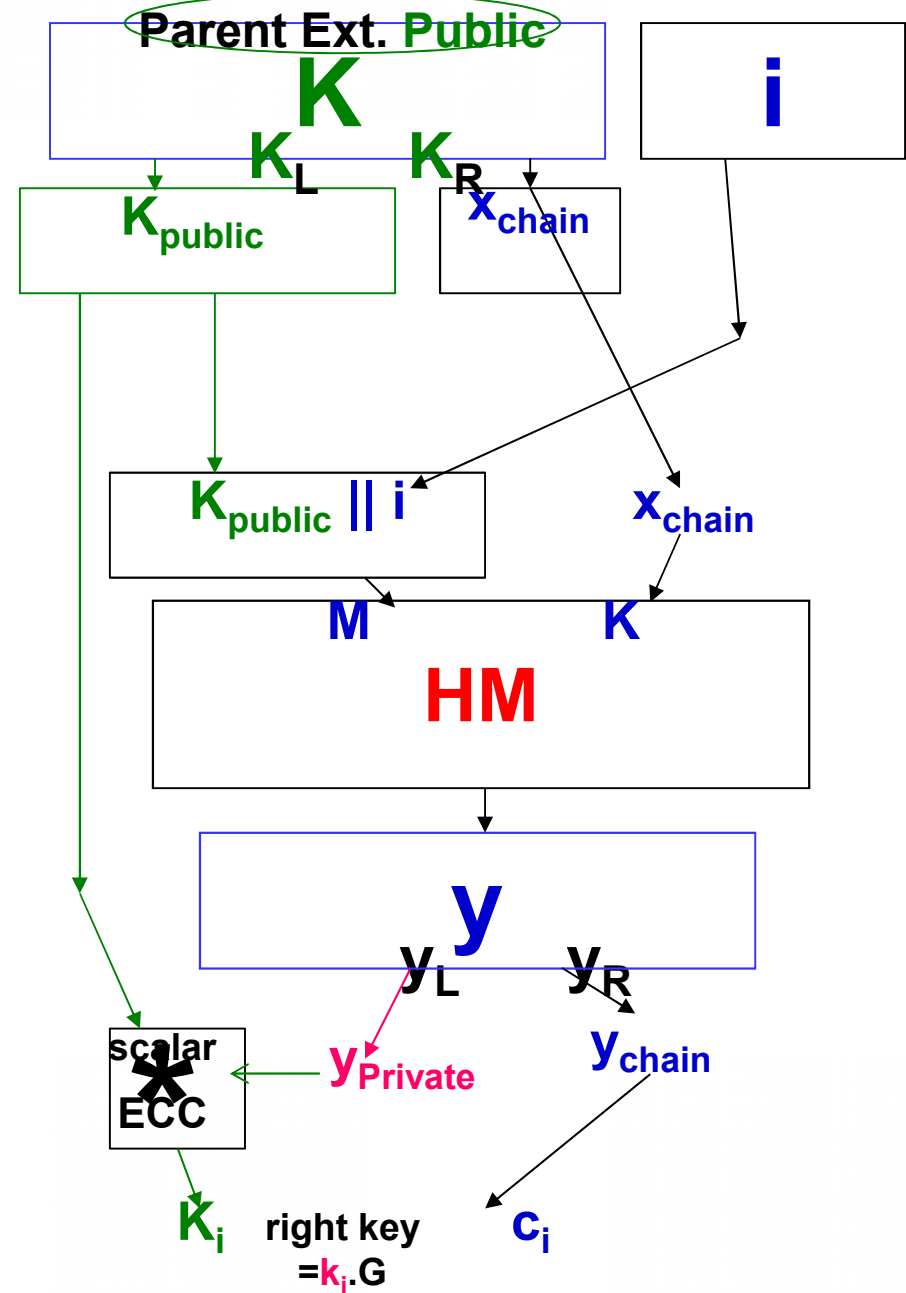
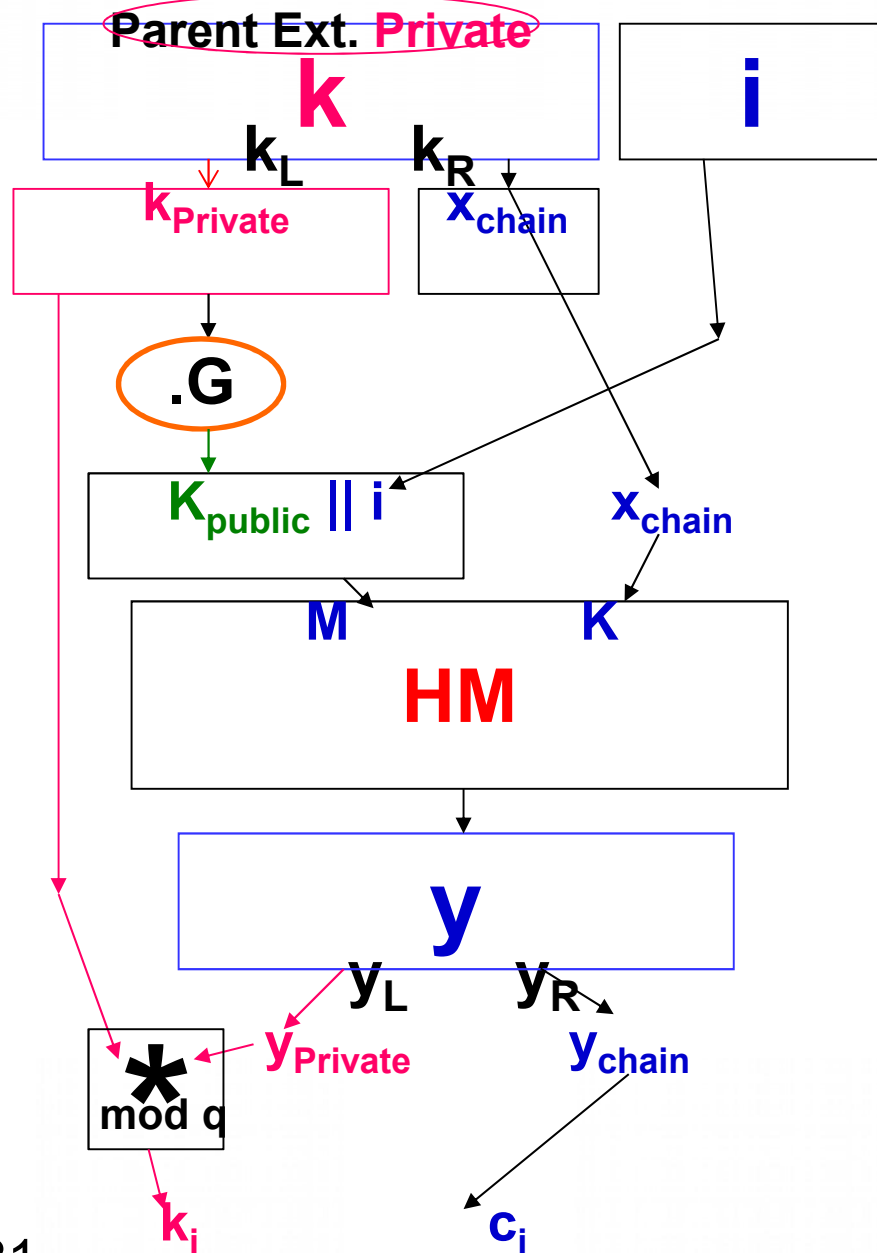
- work across several levels
 - allow audit

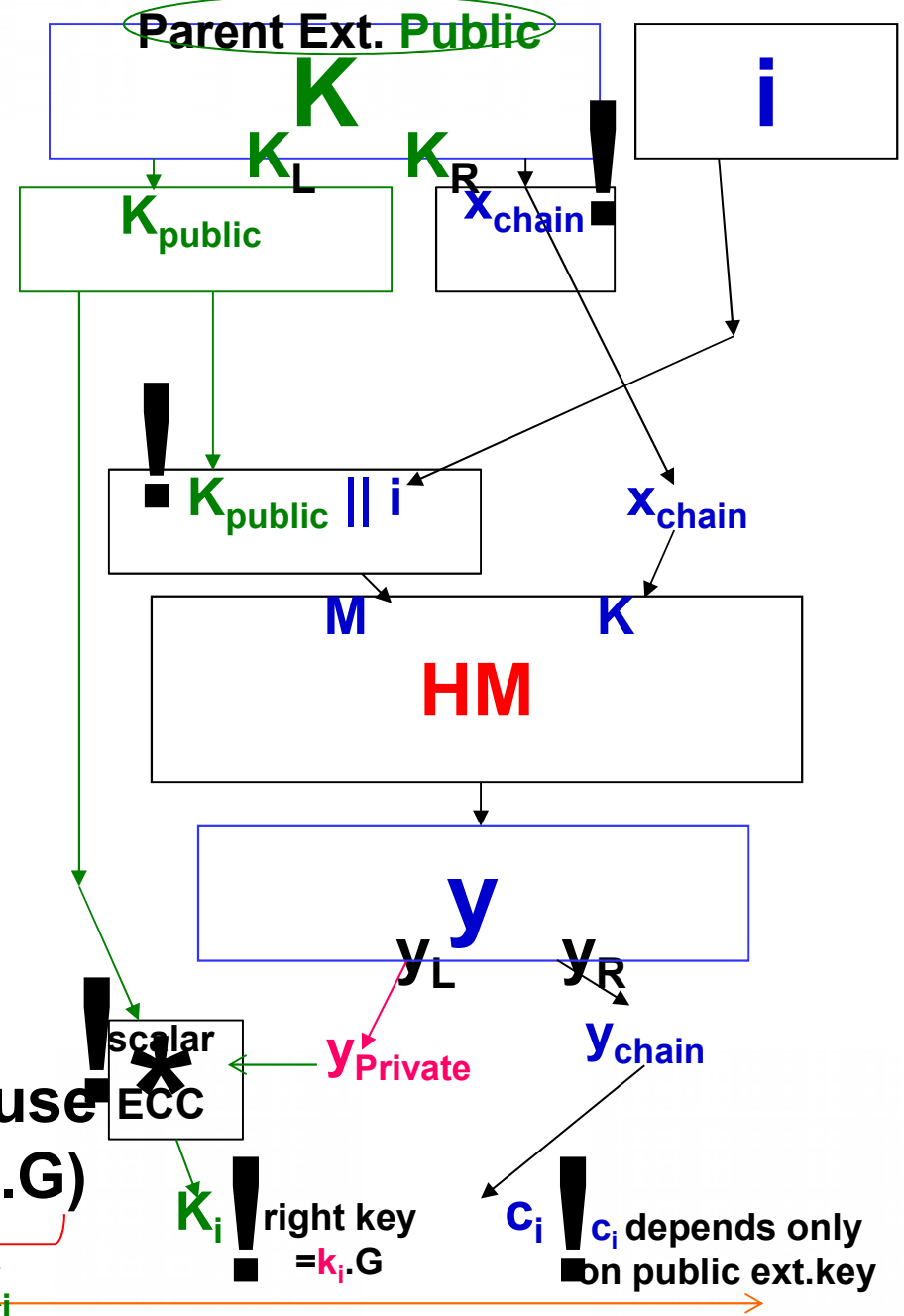
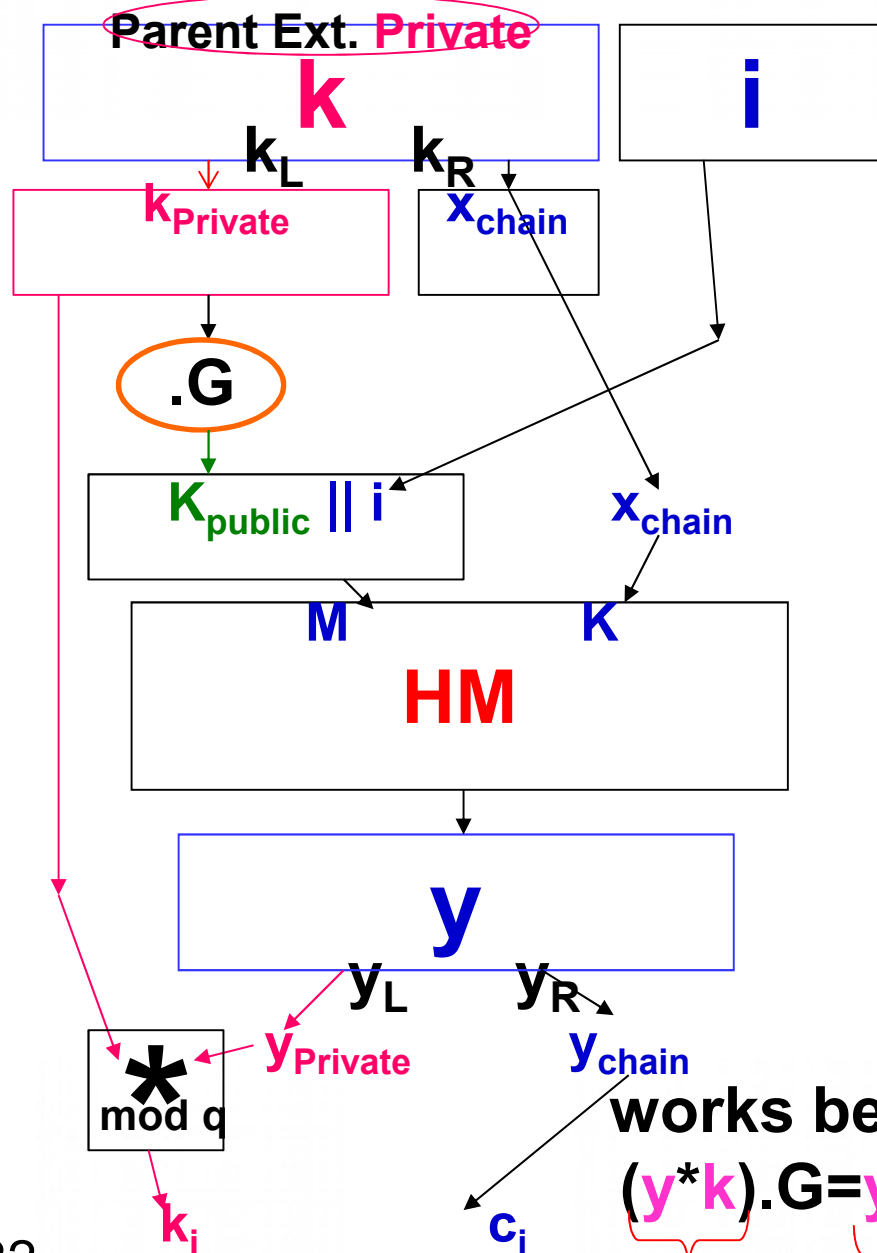
Two Solutions

There are two major “generic” solutions.

1. One uses $\ast \bmod q$ and the fact that $a.(b.G)=(a\ast b).G$
 - older proposal, used before 30 April 2013,
 - probably still used by some software.
2. One uses $+ \bmod q$ AND the homomorphic property of the EC: $a.G+b.G=(a+b).G$.
 - the one which is standardized in BIP032 and massively used TODAY in wallet software, cold storage solutions, bitcoin exchanges, bitcoin payment acceptance and electronic commerce solutions etc..

Not sure that there exist any other solutions
(it is not trivial to design such systems).

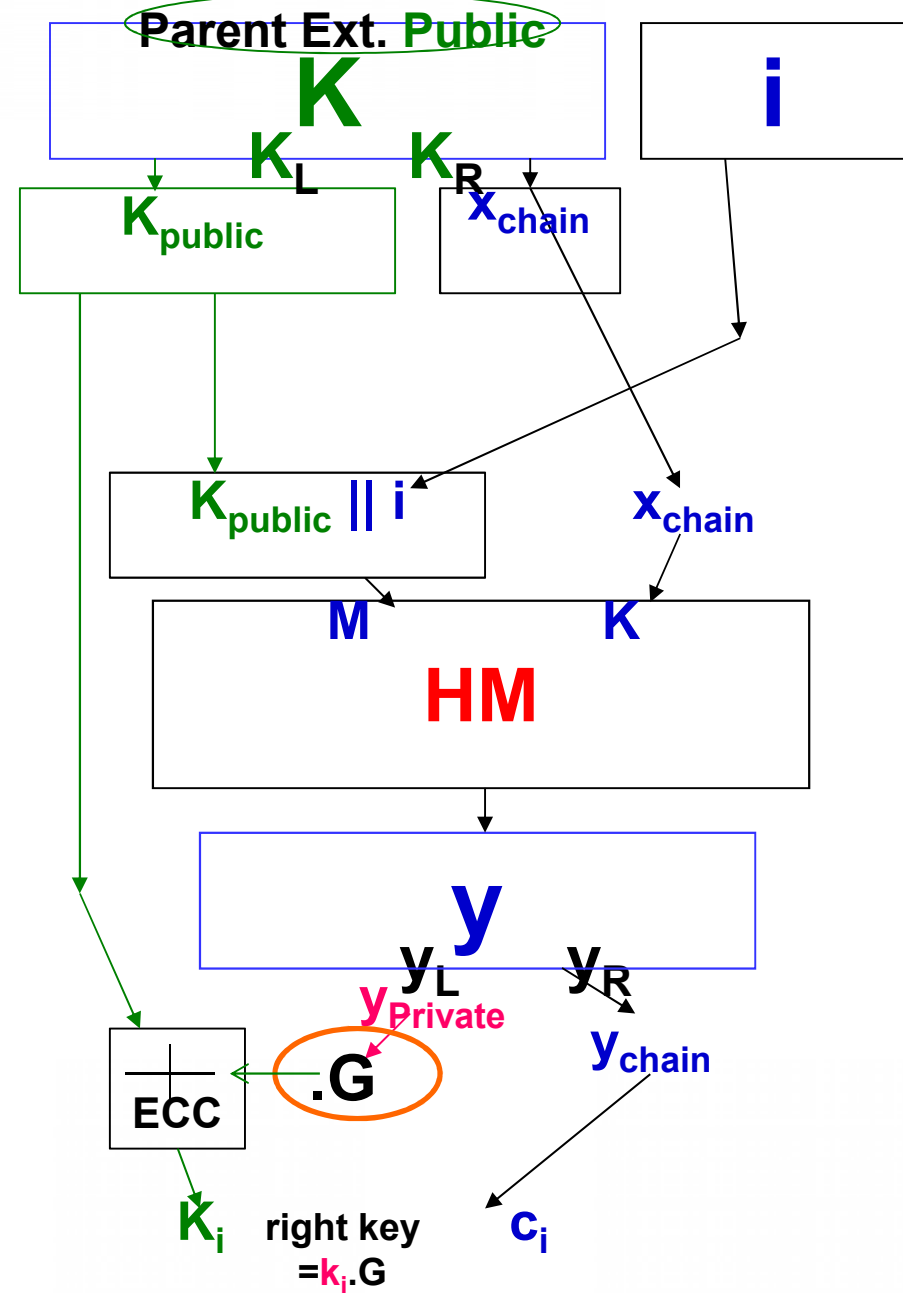
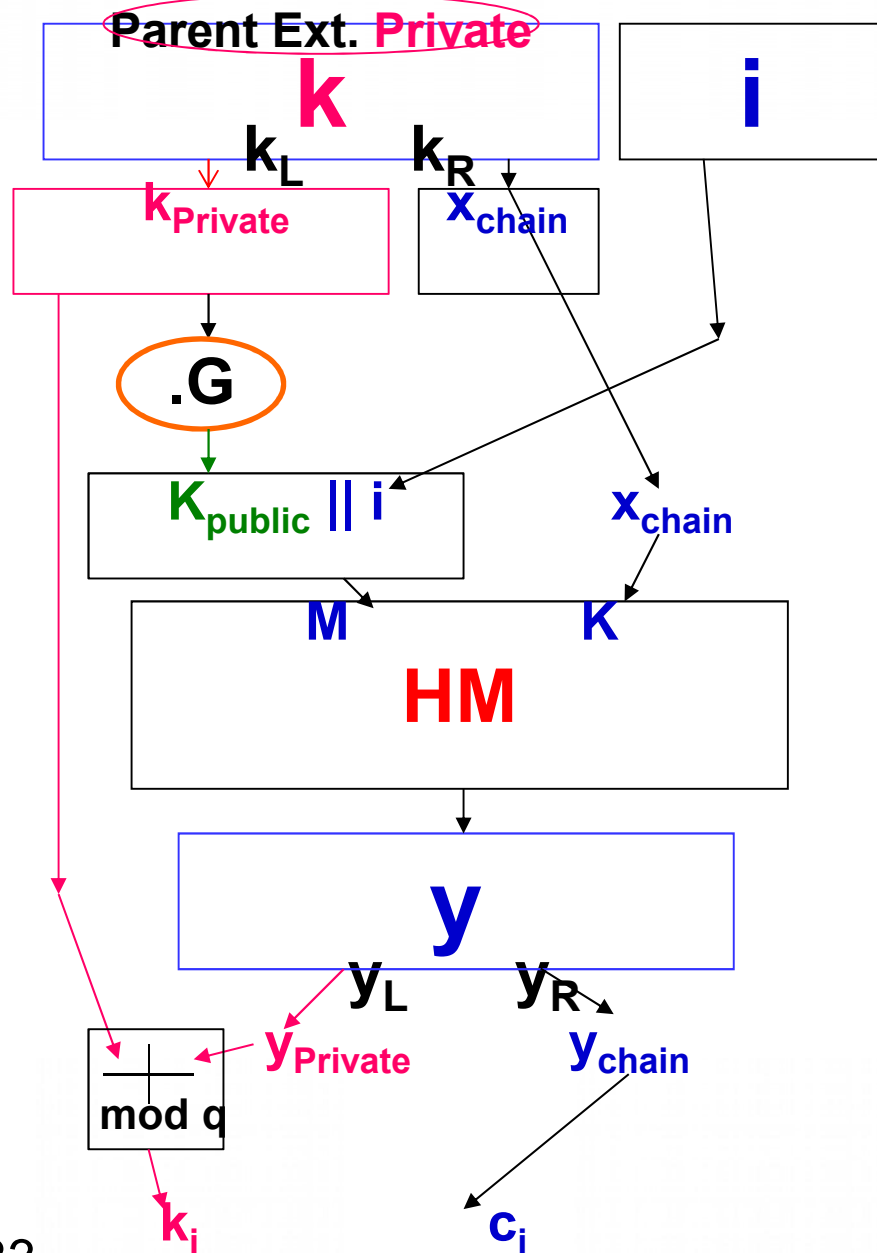


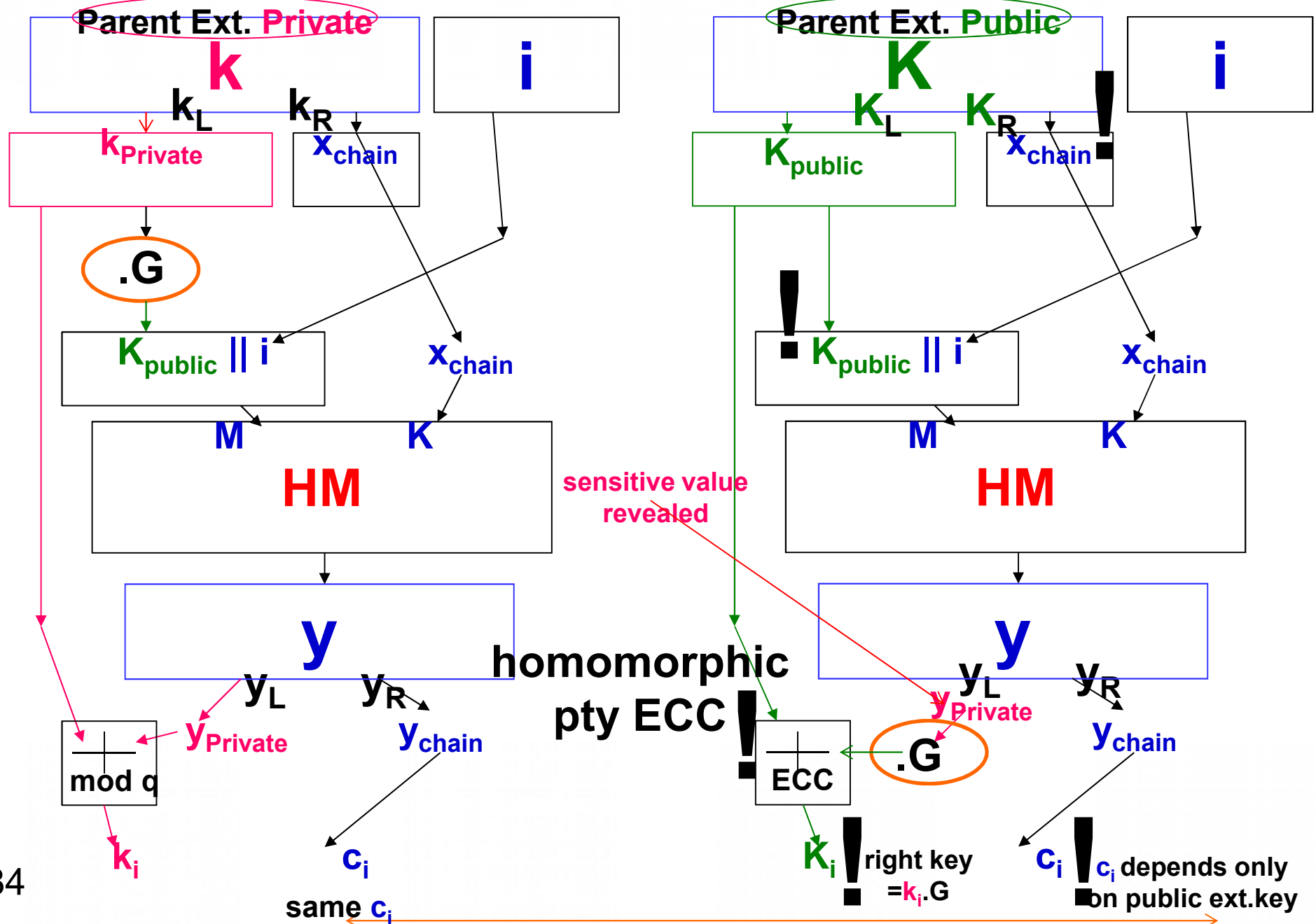


works because

$$(y * k).G = y.(k.G)$$

same c_i k_i K_i





Attacks Sols1/2

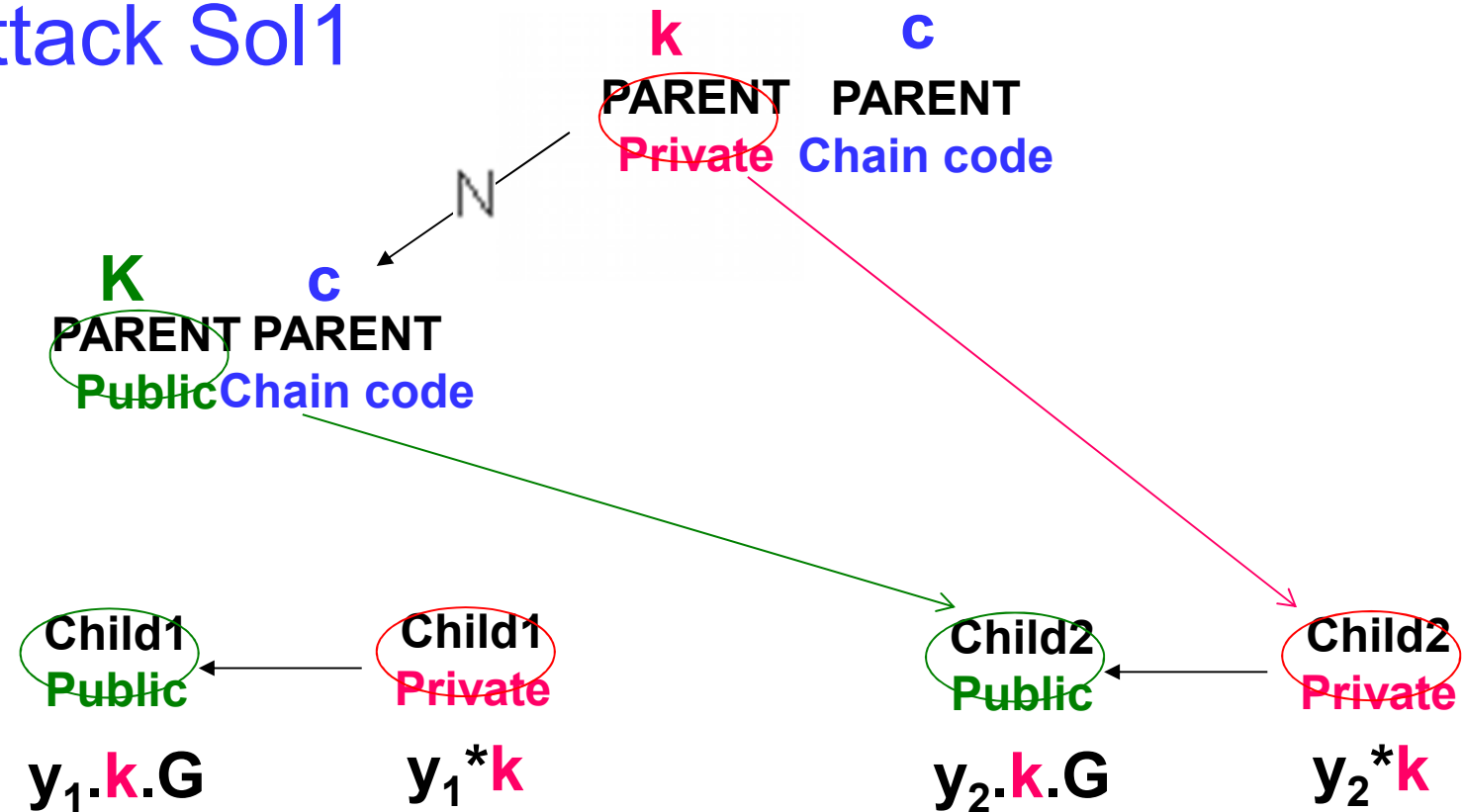
Both have a similar
privilege escalation attack:

Auditor public extended key + 1 private key

=> can compute any other private key with a known
index/address

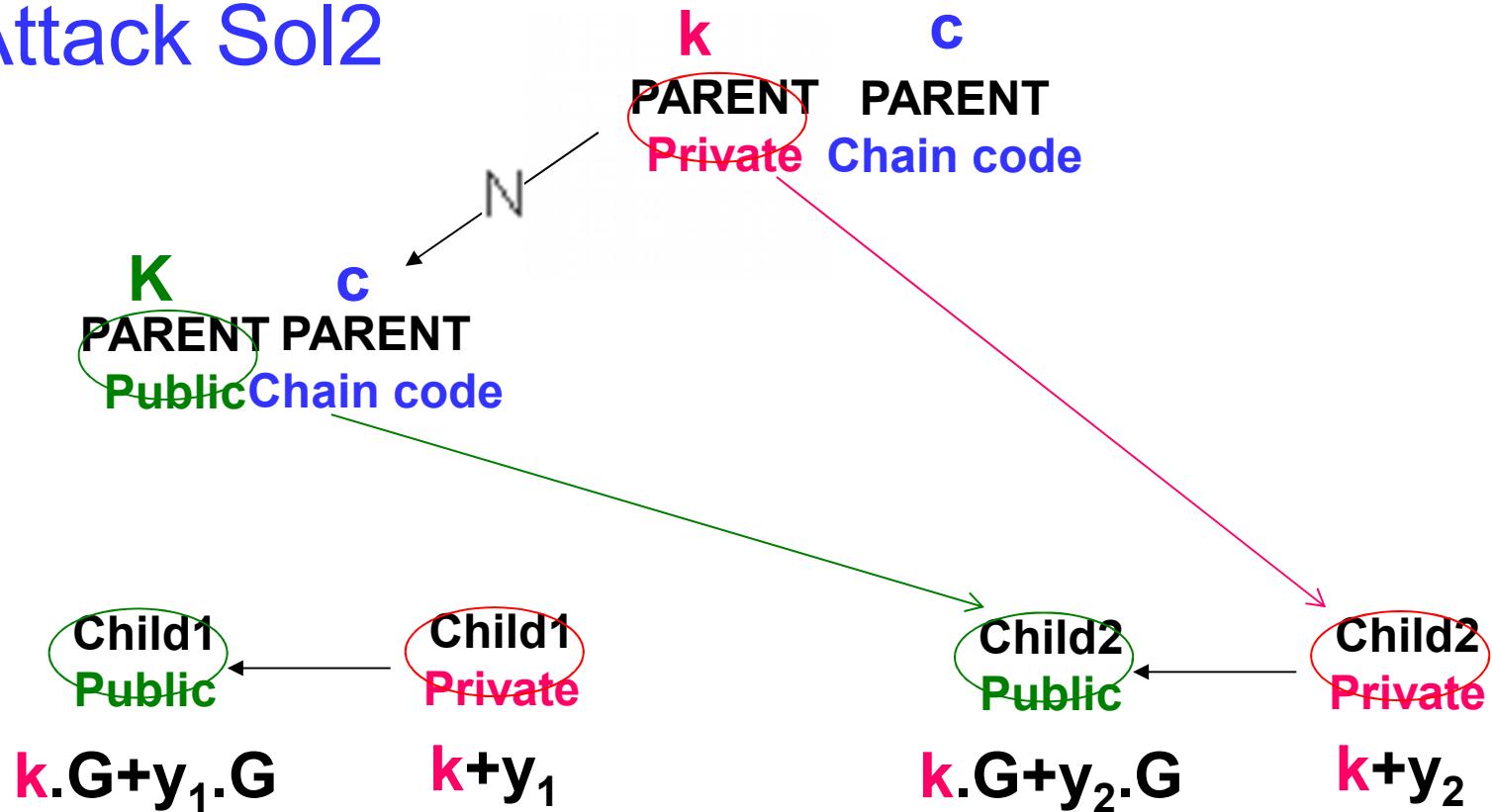
=> can compute ALSO the higher level private key **k**.

*Attack Sol1



Thm: One private key $y_1 \cdot k$ + auditor key
 \Rightarrow compute **MASTER PRIVATE KEY k**
 \Rightarrow any other private key $y_2 \cdot k$.

*Attack Sol2



Thm: One private key $k+y_1$ + auditor key
 \Rightarrow compute the MASTER PRIVATE KEY k
 \Rightarrow any other private key $k+y_2$.

Current Specific Solution BIP032
= Based on Method2

BIP032 Type 2 HD Wallets

More Detailed Study

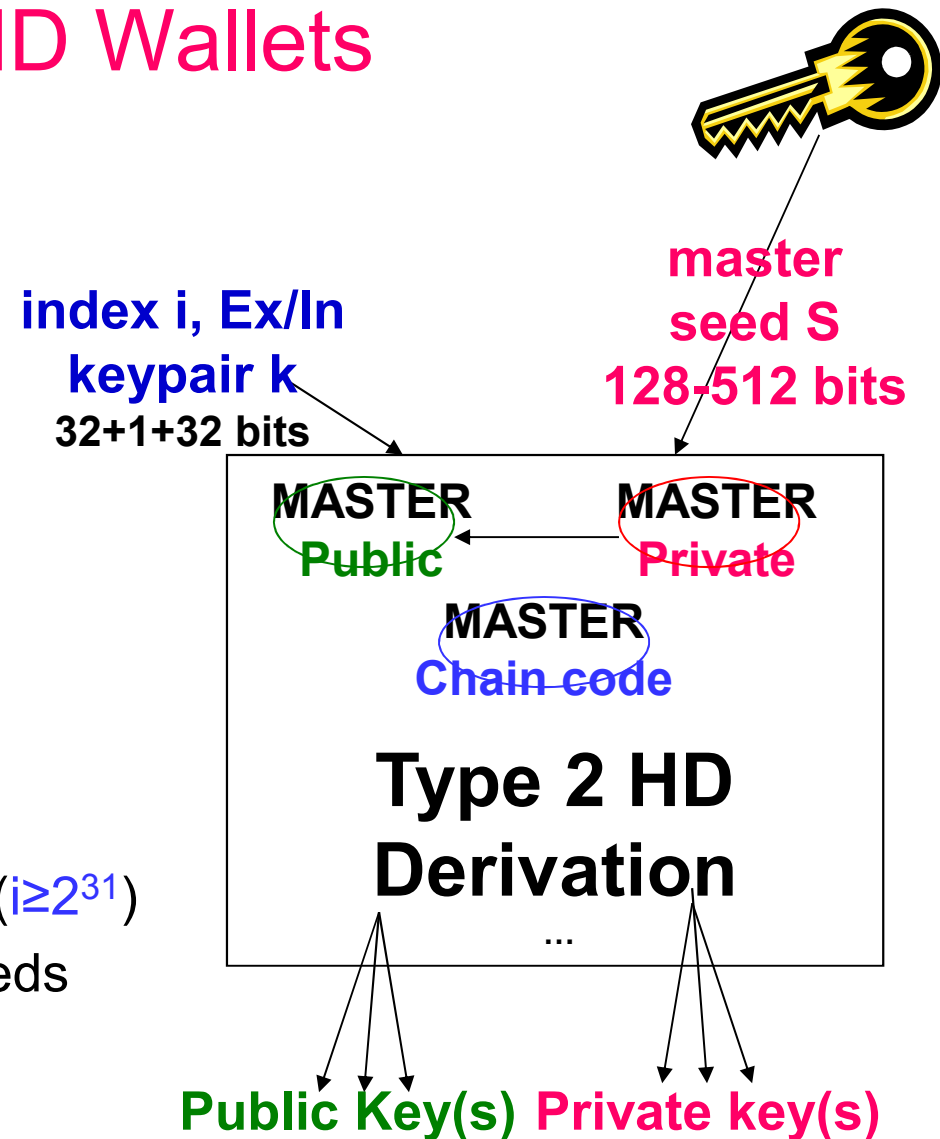
“intended to be shared and used on several systems simultaneously”

Type 2 HD Wallets

Current solution:

BIP0032 [Pieter Wuille]

- At least 4 levels,
- extended secrets: keys on 256
+ chain codes **c** on 256 bits
- multiple key pair chains for each **I**
- 2 versions: normal and hardened ($i \geq 2^{31}$)
- slow hashing protects weaker seeds



CKD functions (there are 3 versions!)

$$\text{CKD}(x,n) = \text{HMAC-SHA512}(x_{\text{Chain}}, x_{\text{PubKey}} \parallel n)$$

HMAC-SHA512

Goal: avoids $H(m||key)$ constructions
(which are subject to certain specific attacks)

Definition (from [RFC 2104](#) [\[edit\]](#))

$$HMAC(K, m) = H((K \oplus opad) | H((K \oplus ipad) | m))$$

where

H is a cryptographic hash function,

K is a secret key padded to the right with extra zeros to the input block size of the hash function, or the hash of the original key if it's longer than that block size,

m is the message to be authenticated,

$|$ denotes concatenation,

\oplus denotes exclusive or (XOR),

$opad$ is the outer padding (0x5c5c5c...5c5c, one-block-long hexadecimal constant),

and $ipad$ is the inner padding (0x363636...3636, one-block-long hexadecimal constant).

1024

HMAC-SHA512 = separate RFC 4231

Hashes twice with a key.

Definition (from [RFC 2104](#))

$$HMAC(K, m) = H((K \oplus opad) || H((K \oplus ipad) || m))$$

where

H is a cryptographic hash function,

K is a secret key padded to the right with the hash of the original key if it's longer than

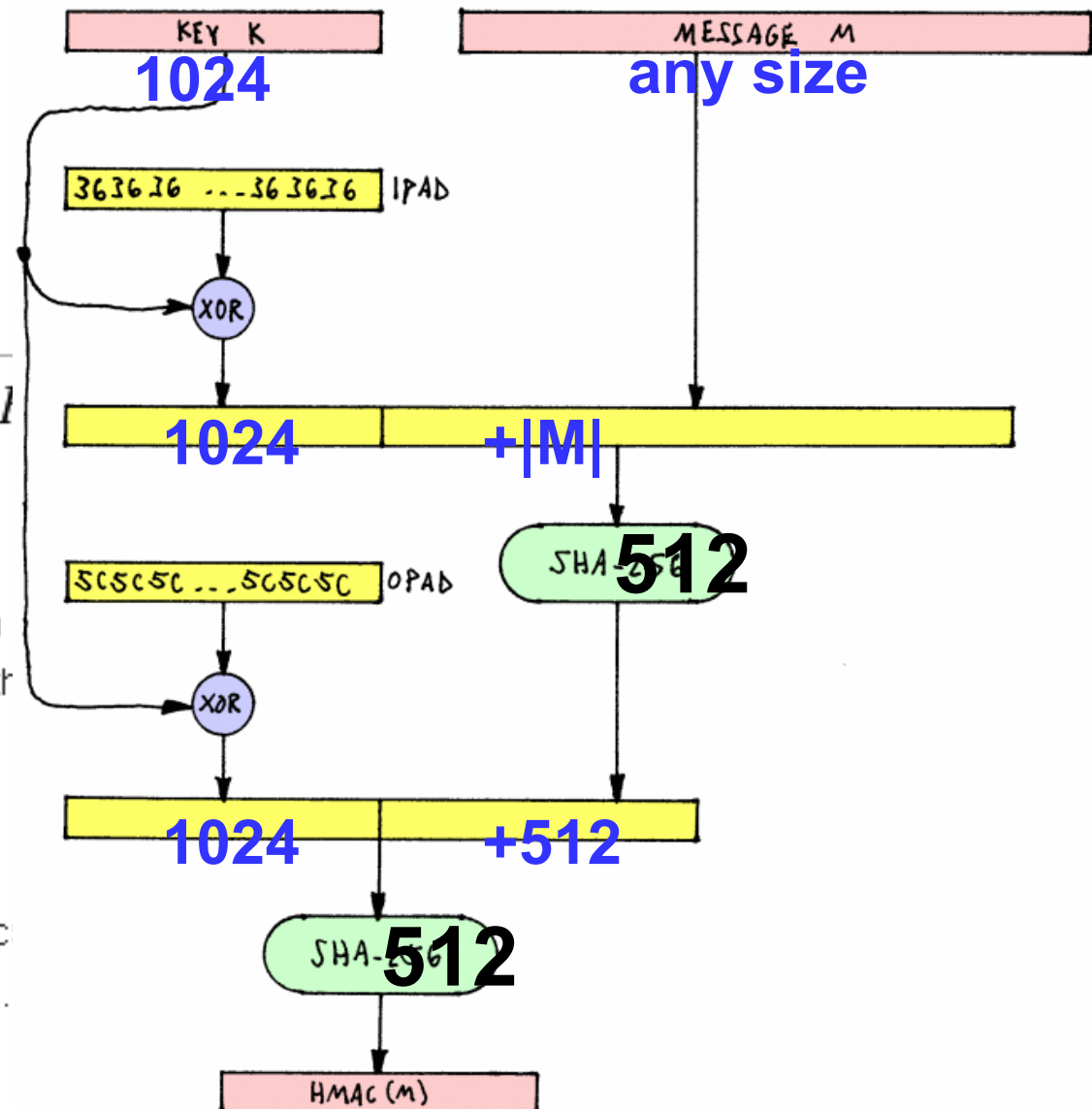
m is the message to be authenticated,

$||$ denotes concatenation,

\oplus denotes exclusive or (XOR),

$opad$ is the outer padding (0x5c5c5c...5c

and $ipad$ is the inner padding (0x363636...

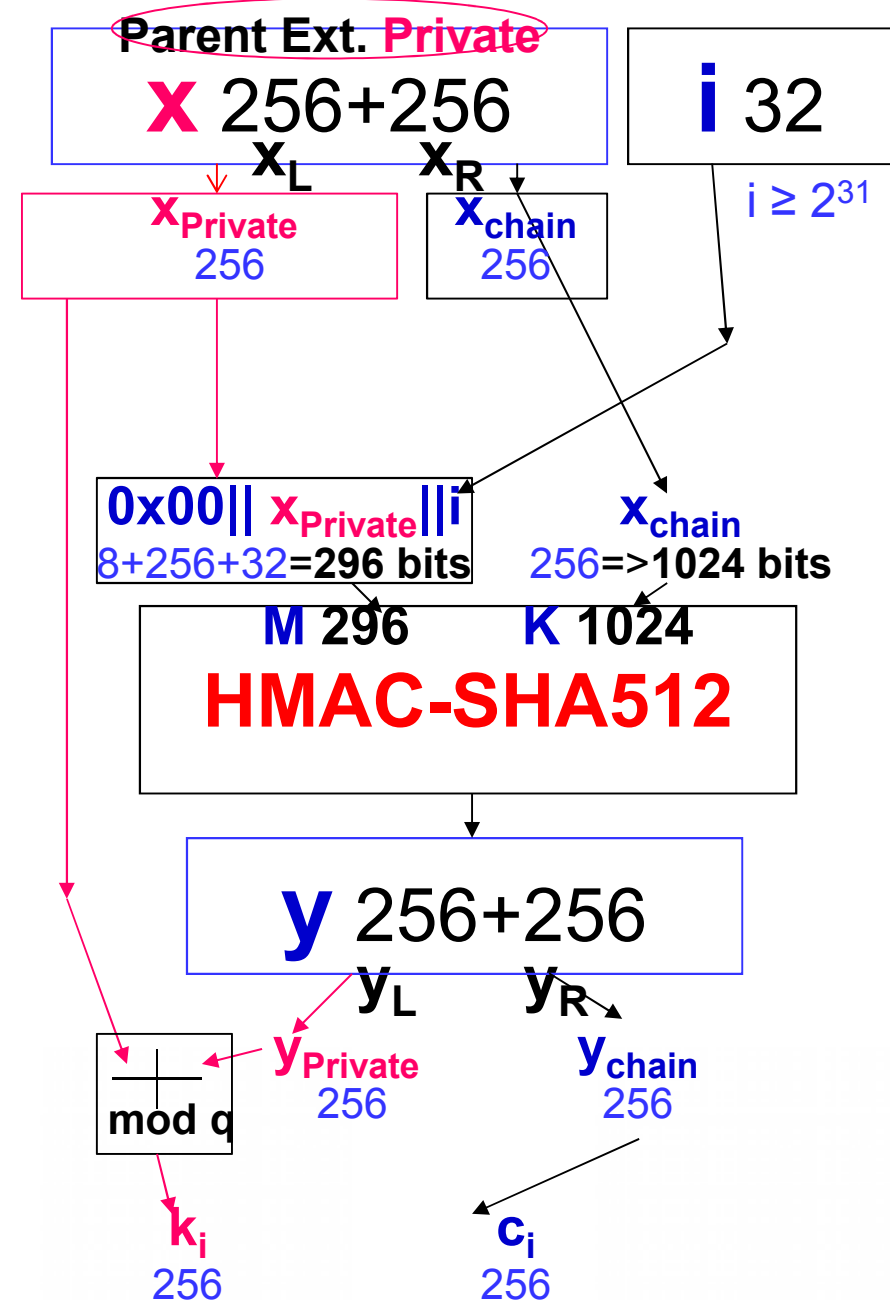


HMAC-SHA512 usage:
 $CDKpriv_H$ (hardened $i \geq 2^{31}$)

$CKD(x,n) =$

hardened

disables “homomorphic magic”
 decreases the scope of “audit”



HMAC-SHA512 usage: CDKpriv_N (not hardened $i < 2^{31}$)

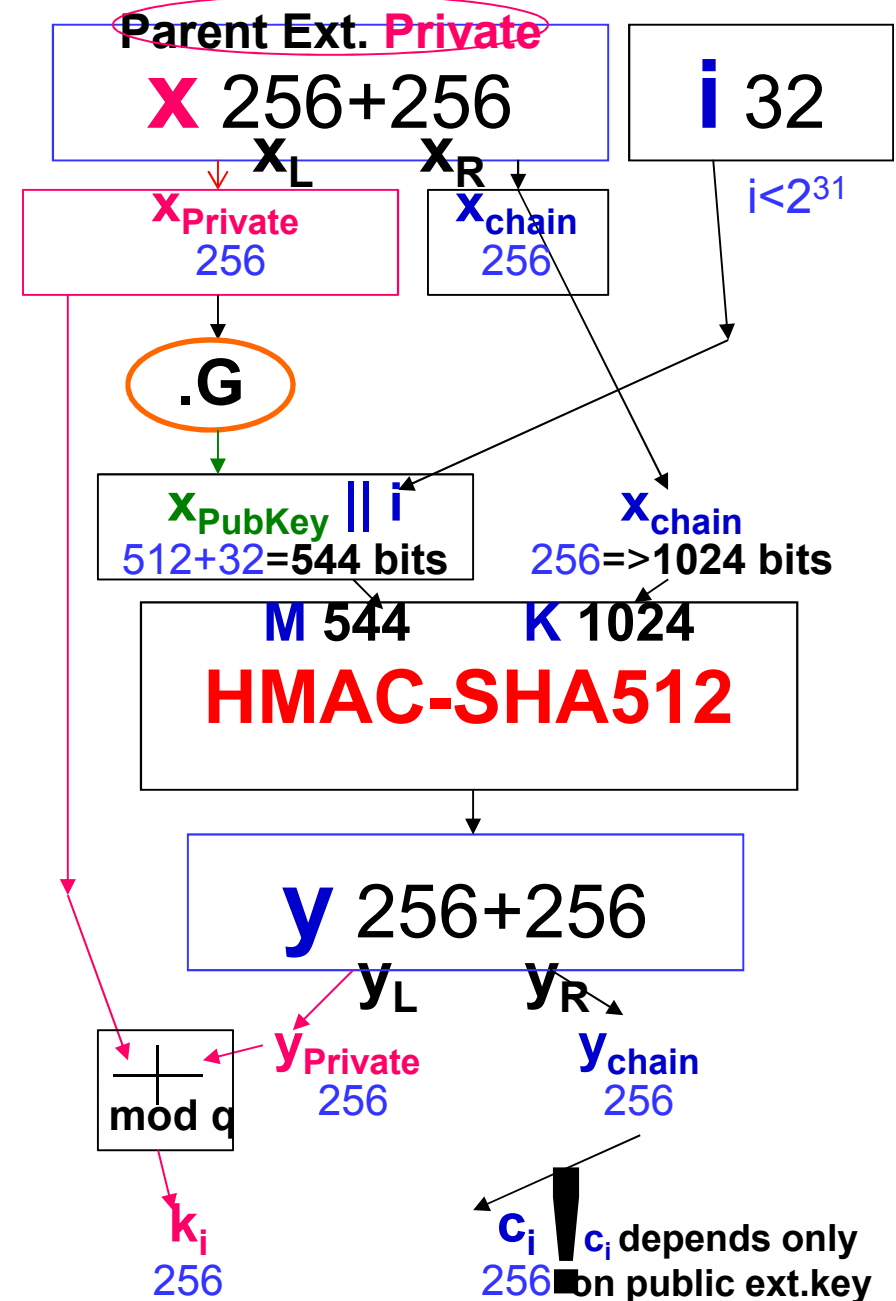
Child Key Derivation Function

$$\text{CKD}(x, n) =$$

$$\text{HMAC-SHA512}(x_{\text{Chain}}, x_{\text{PubKey}} \parallel n)$$

derived

$y = \text{HMAC-SHA512}(\text{Key} = x_{\text{chain}}, \text{Data} = x_{\text{PubKey}} \parallel i)$
 y has 512 bits = 2×256 bits

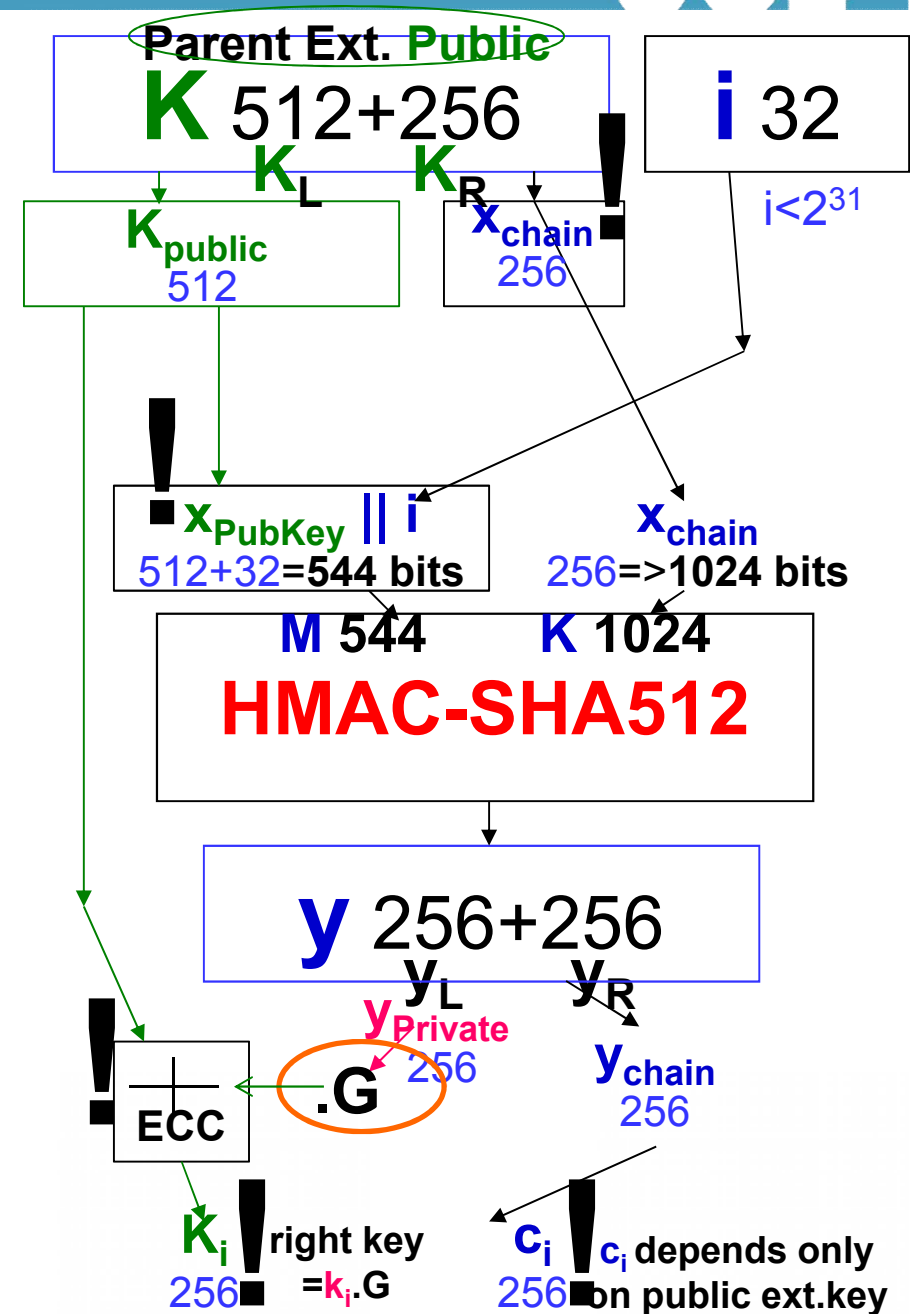


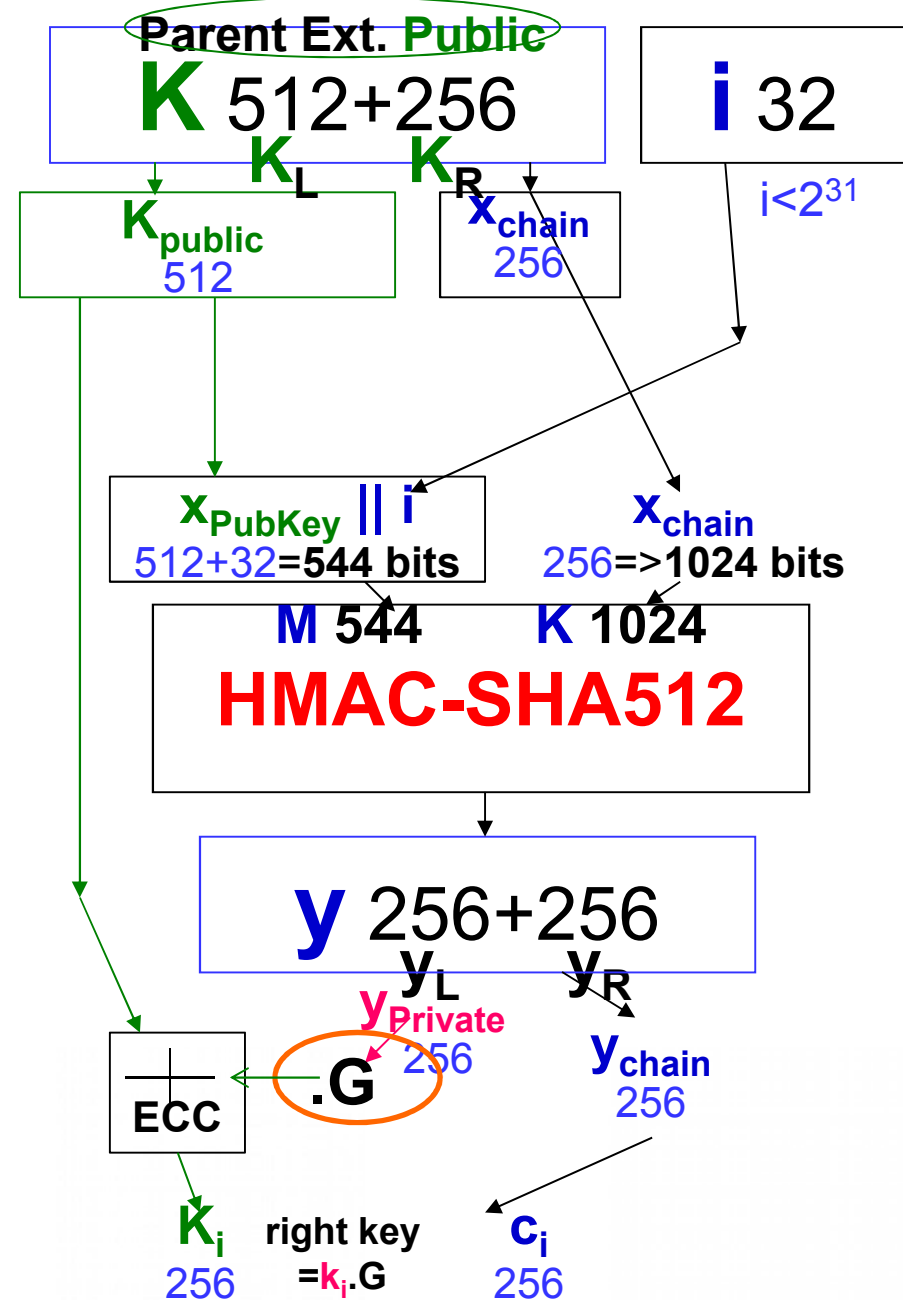
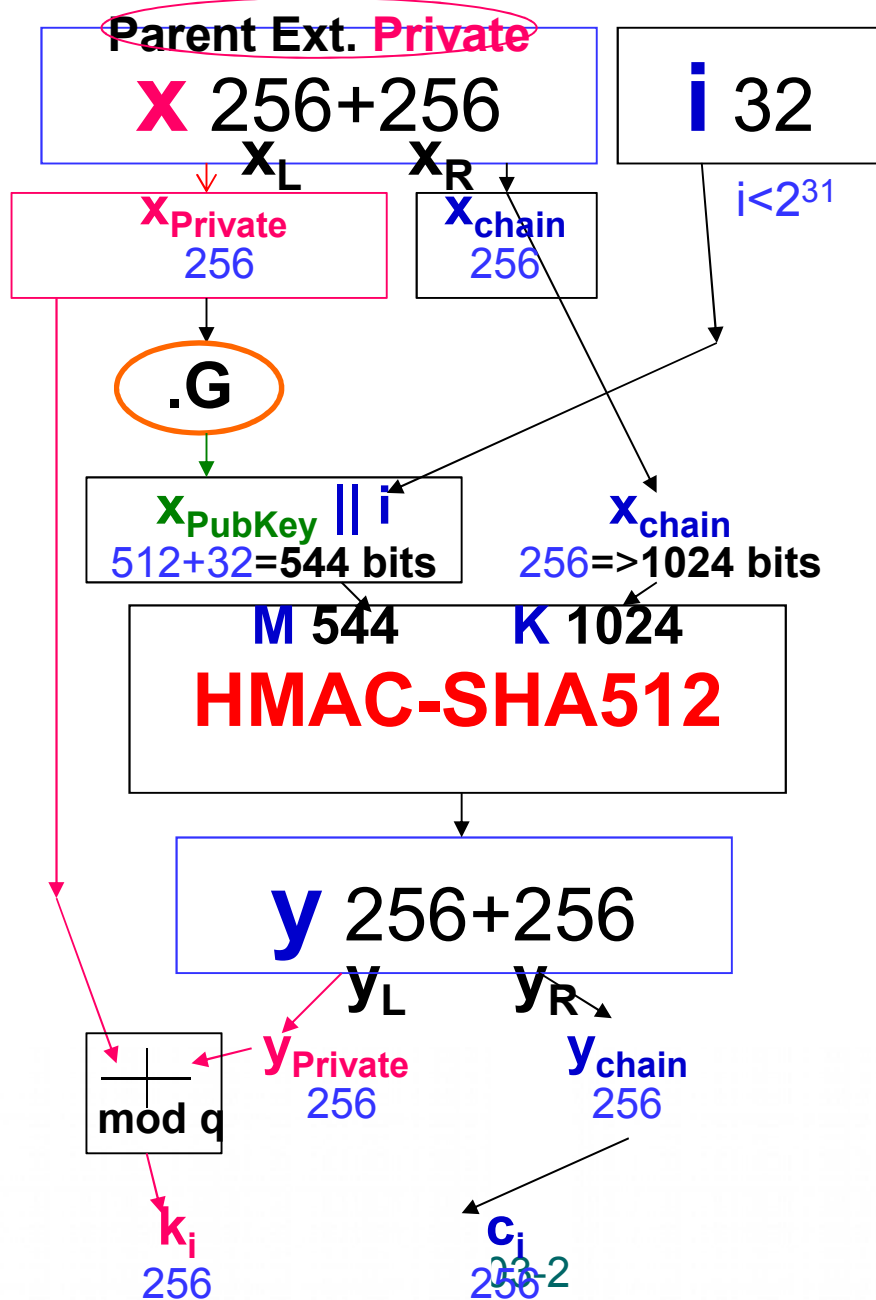
HMAC-SHA512 usage: CDKpub not hardened ONLY $i < 2^{31}$

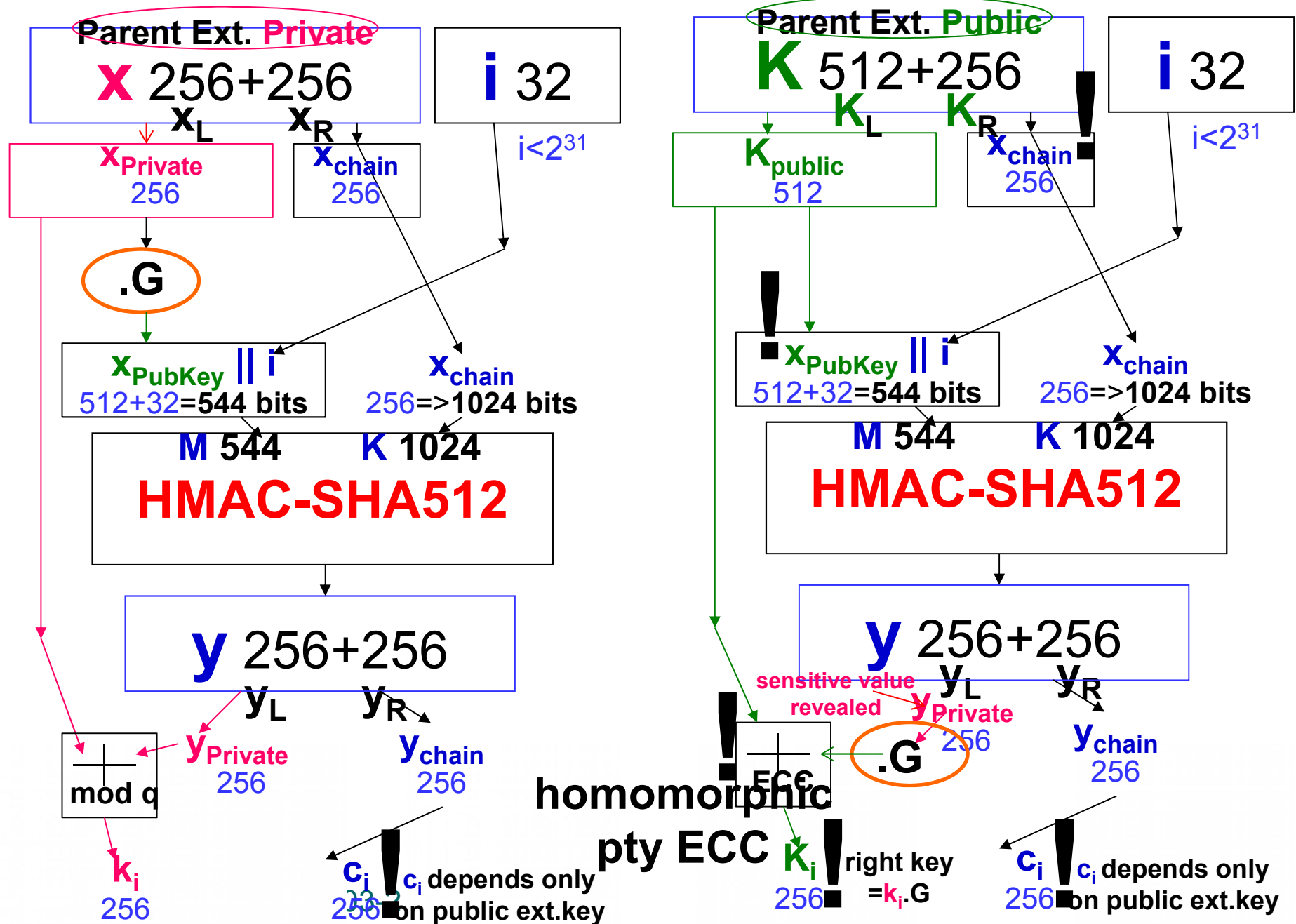
$$\text{CKD}(x, n) = \text{HMAC-SHA512}(x_{\text{Chain}}, x_{\text{PubKey}} \parallel n)$$

direct

$y = \text{HMAC-SHA512}(\text{Key} = x_{\text{chain}}, \text{Data} = x_{\text{PubKey}} \parallel i)$
 y has 512 bits = 2×256 bits

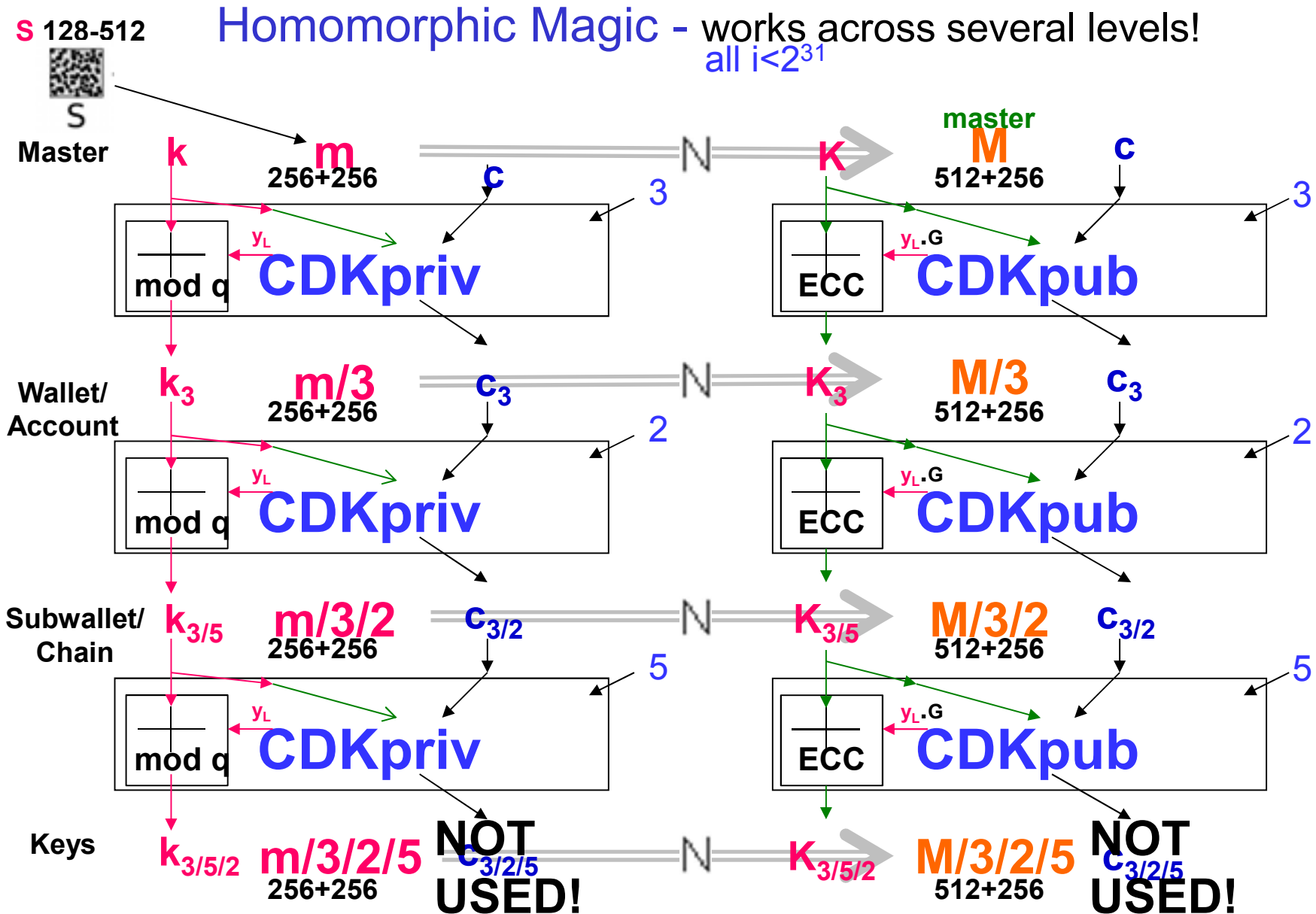




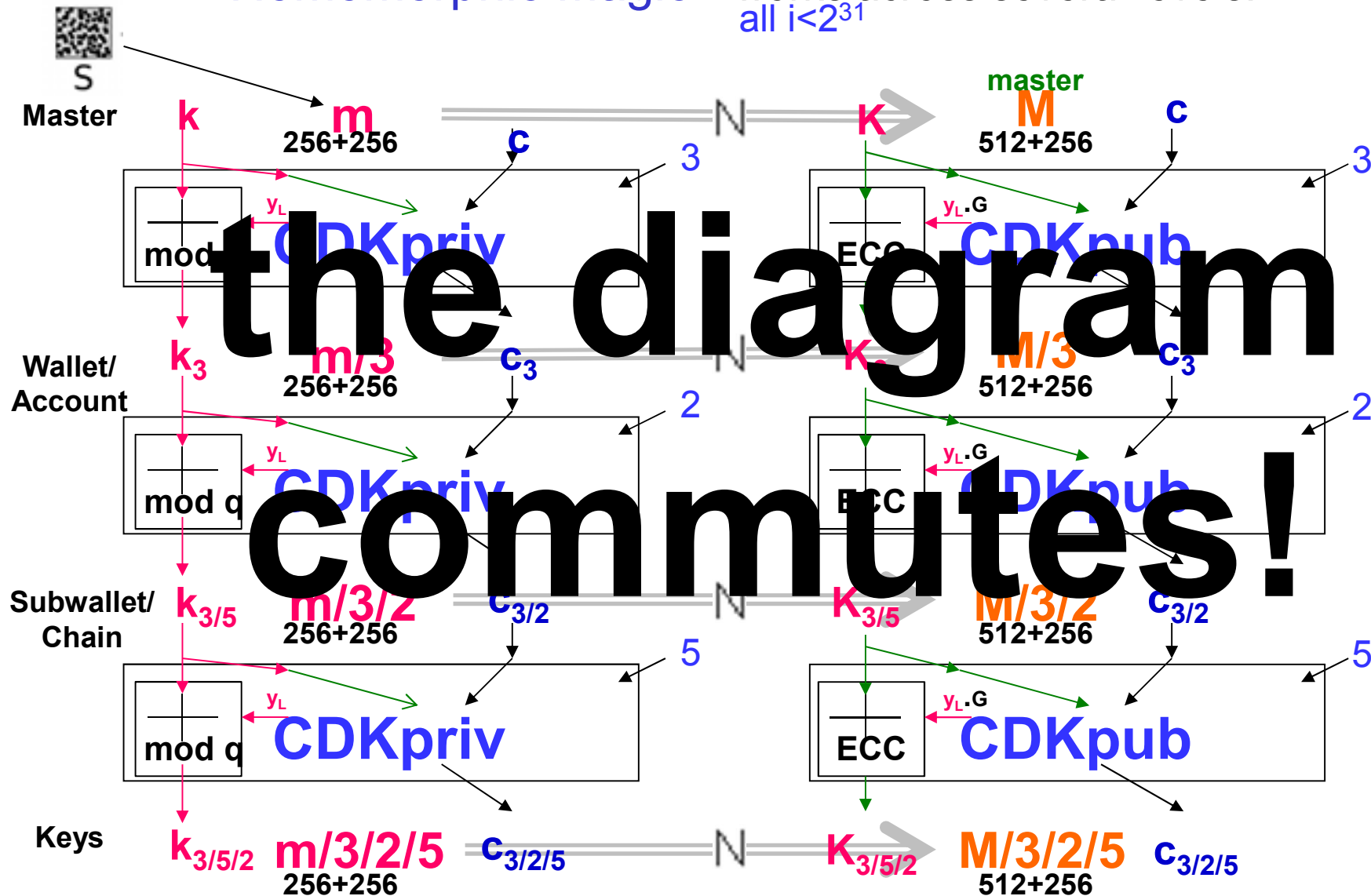


BIP032

Working Principle



Homomorphic Magic - works across several levels!
all $i < 2^{31}$



BIP032

Calculus

(remarkable identities)

Several Ways

Many extended public keys 512+256
have several ways to compute them!

$$N(m/a/b/c) = N(m/a/b)/c = N(m/a)/b/c = \\ N(m)/a/b/c = M/a/b/c$$

Effect of hardening:

$$N(m/a_H/b/c) = N(m/a_H/b)/c = N(m/a_H)/b/c.$$

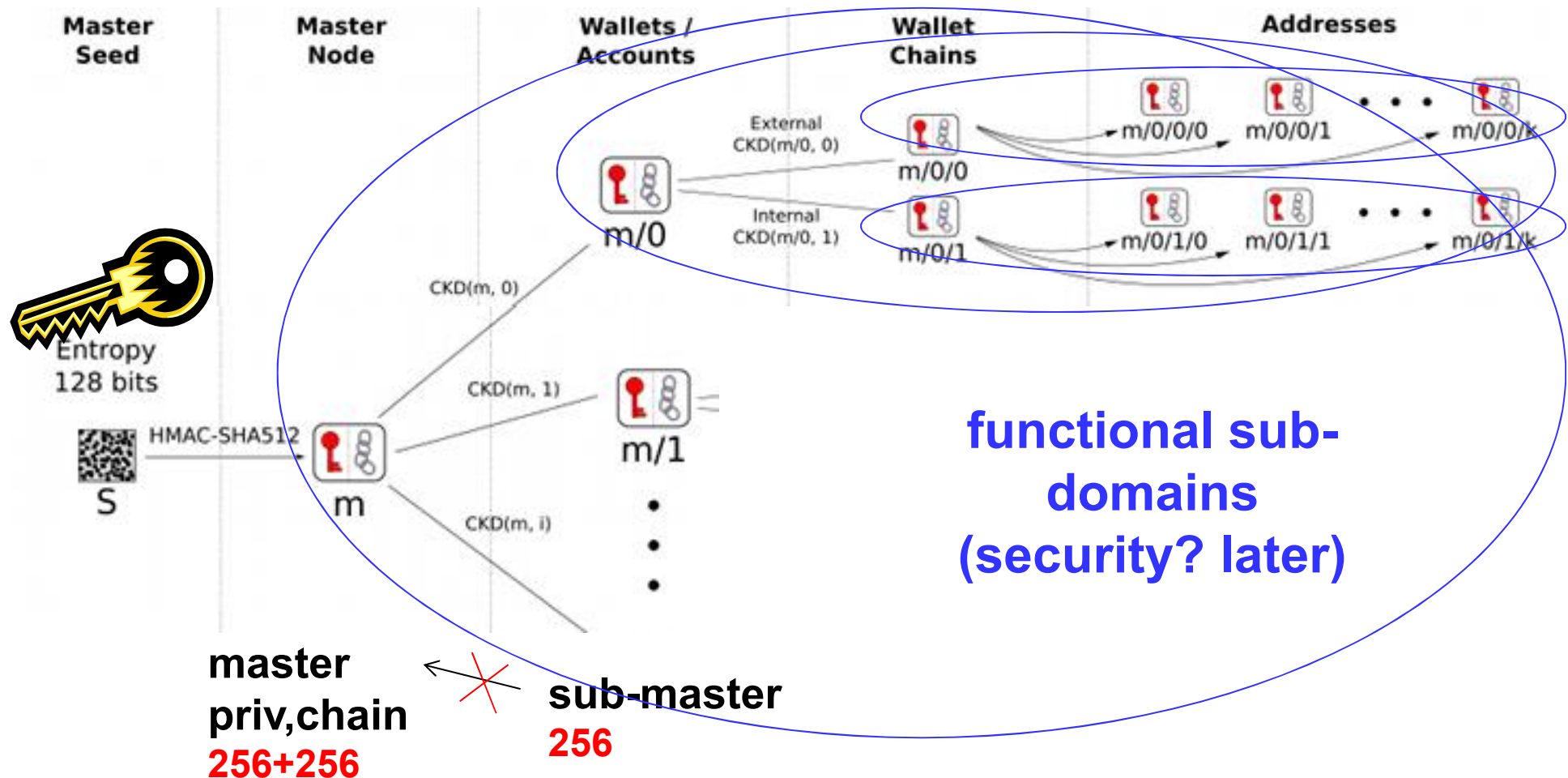
BUT

$N(m)/a_H$ is NOT defined!

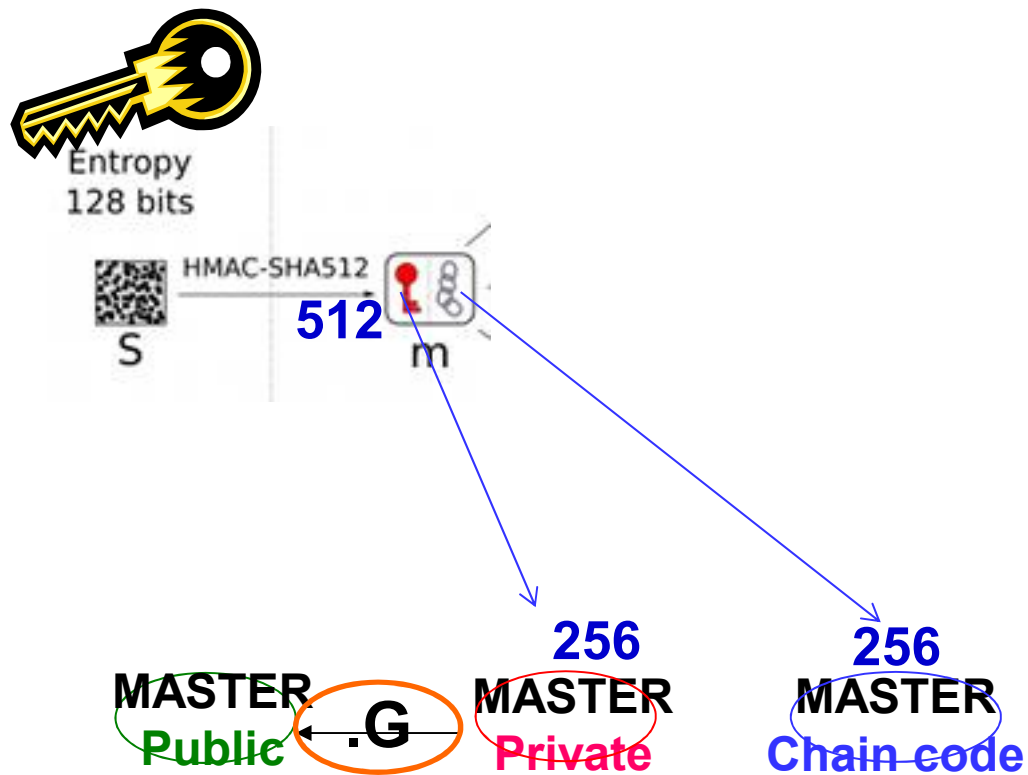
BIP032

Details

BIP032 – 4 Levels

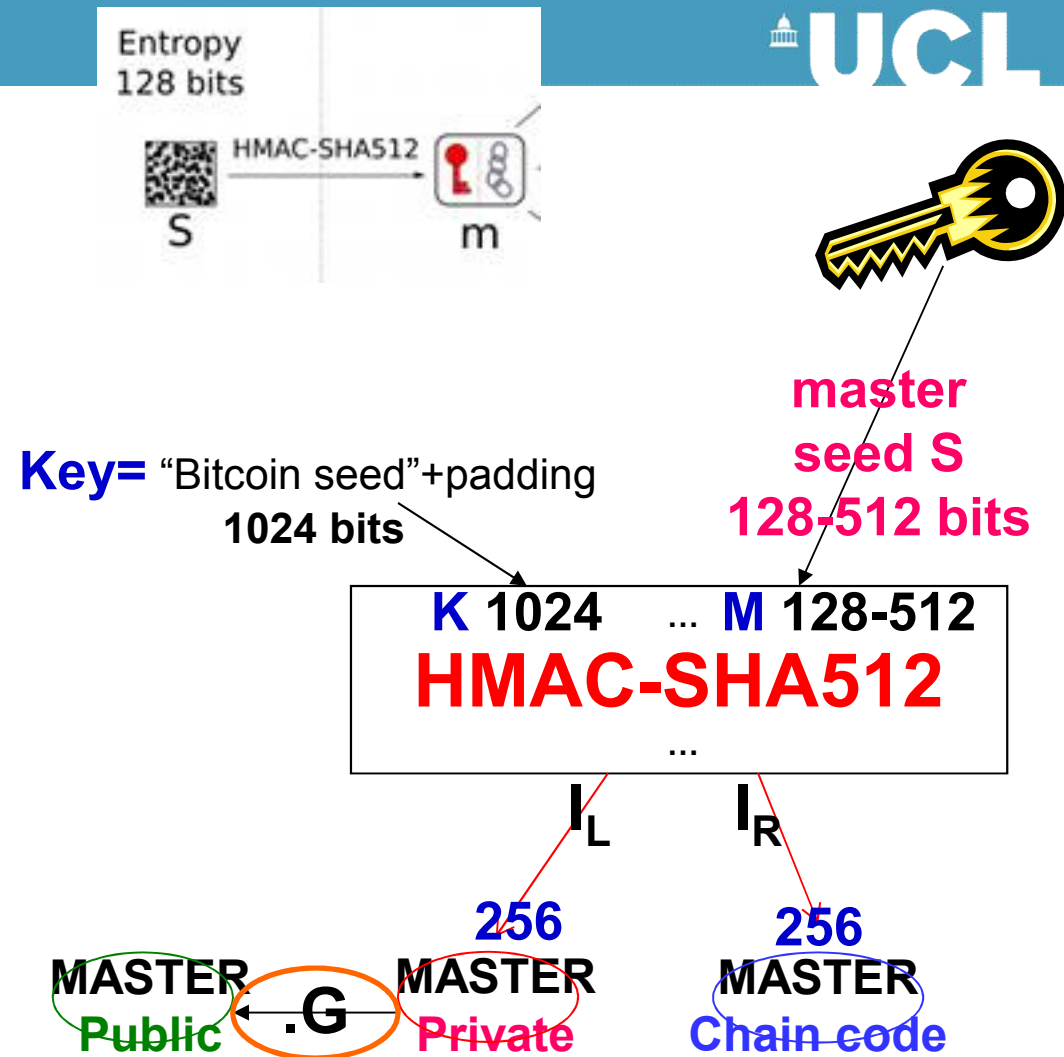


Step1



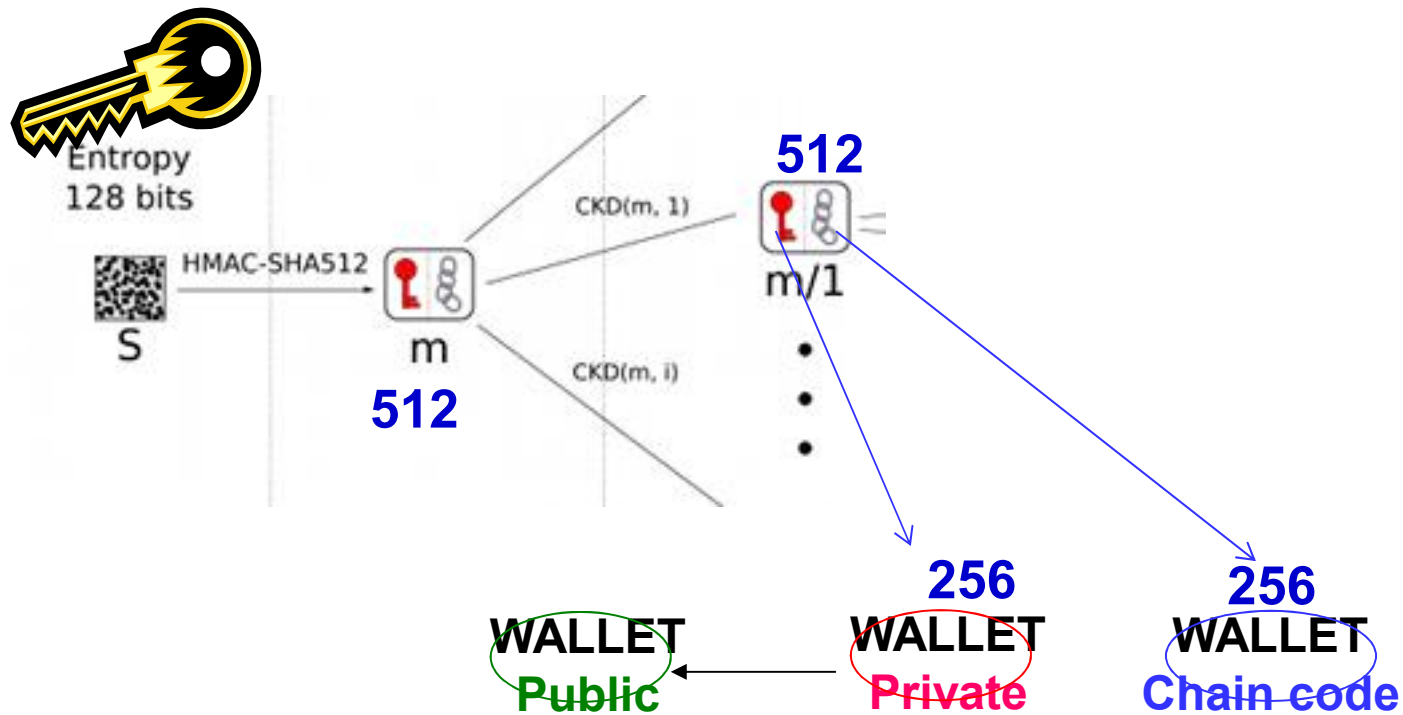
*Step 1

Q1: HMAC not needed (key=cst)
but seems correctly used
(a hash function would be OK)
Q2: this HMAC is a OWF!



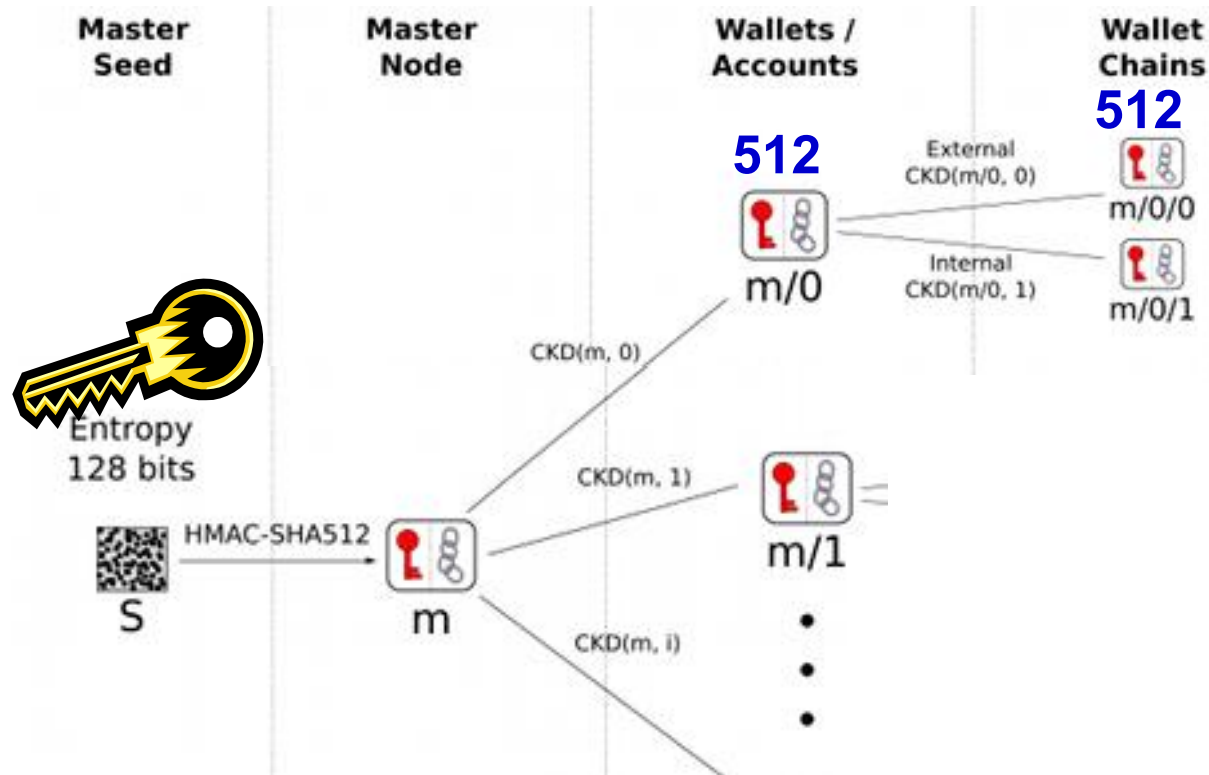
$I = \text{HMAC-SHA512}(\text{Key}=\text{"Bitcoin seed"}, \text{Data}=S),$
512 bits = 2*256 bits

Step 2

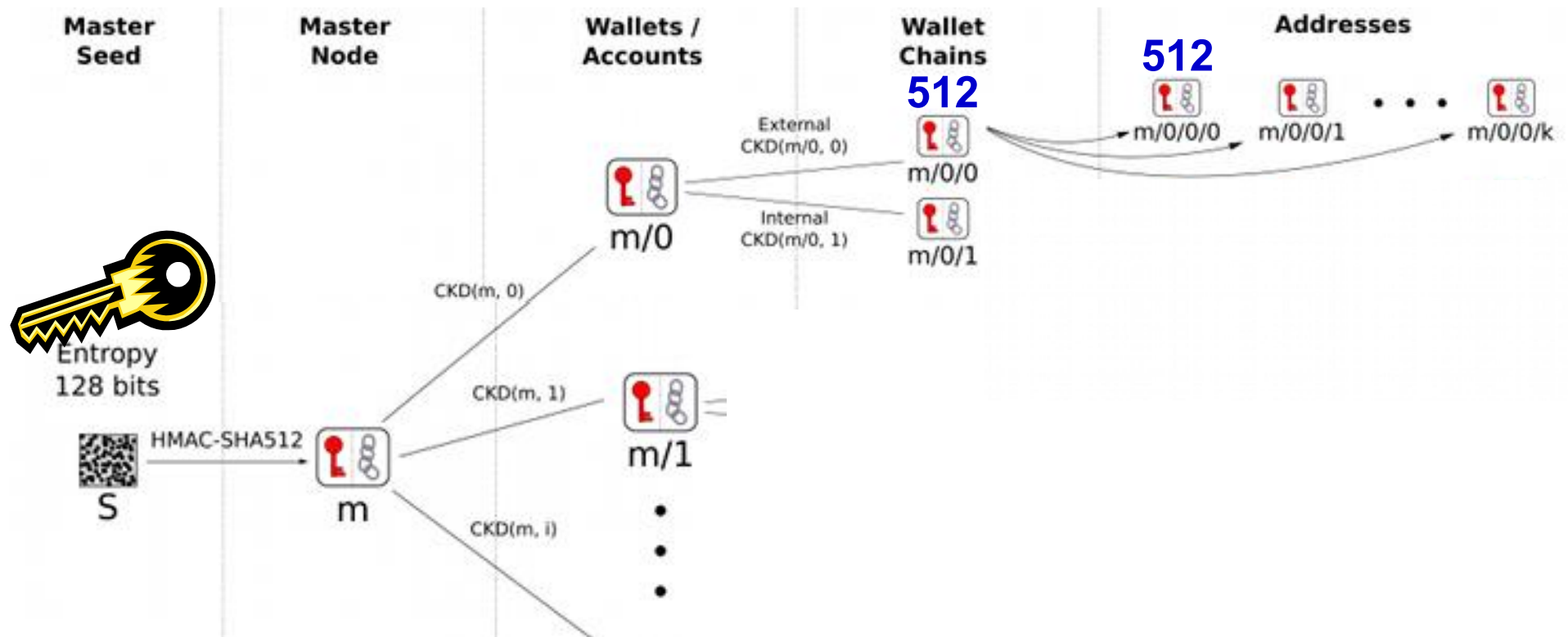


CAN also be derived from
higher-level **Public** ext. Keys!

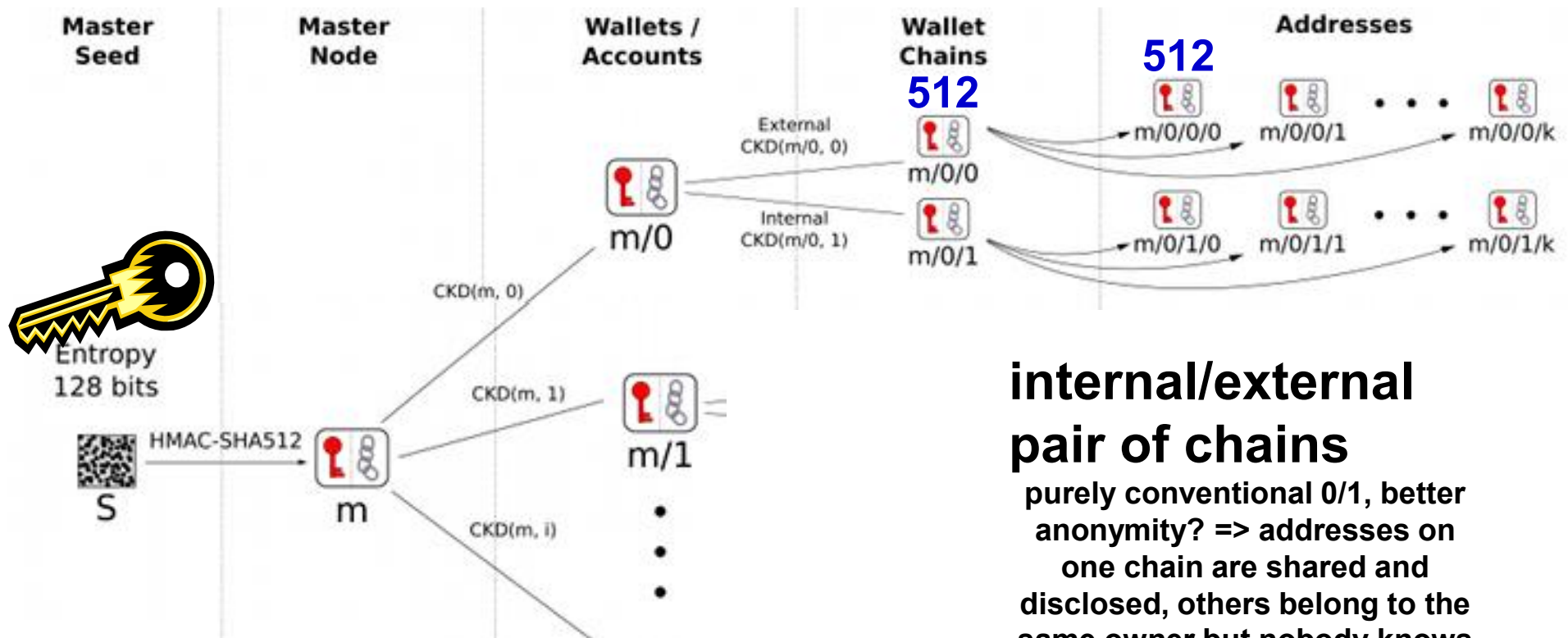
Step 3



Step 4



Step 4



internal/external pair of chains

purely conventional 0/1, better anonymity? => addresses on one chain are shared and disclosed, others belong to the same owner but nobody knows it... => can return moneys to ourselves on secret addresses...

BIP032

Practical Example

Installing ku

1. Download and install Python 2.7 or 3.4, e.g.
<https://www.python.org/ftp/python/3.4.1/python-3.4.1.msi> for Win
*Under windows a working cygwin/gcc or Visual Studio nmake MIGHT be needed? Not sure. Probably not.
2. Install package setuptools:
 - Windows: Run https://bootstrap.pypa.io/ez_setup.py
 - Unix: `wget https://bootstrap.pypa.io/ez_setup.py -O - | sudo python`
3. Download pycoin: <https://github.com/richardkiss/pycoin/archive/master.zip>
4. Unpack in one directory (under W7 avoid protected dirs like Program Files)
5. Now sequence of build and install:
 - Win: `setup.py build`
 - `setup.py install`
 - Unix: `sudo python setup.py build`
 - `sudo python setup.py install`
6. cd to sub-directory `.\build\lib\pycoin\scripts`
7. To run type `gewallet.py` or `ku.py`
 1. or `python ku.py` or just `ku` depending on OS..

Extended Keys - Example

Uses Pycoin lib by Richard Kiss

>ku P:foo => produces two 111-digit base58 strings!!!!

wallet key:

xprv9s21ZrQH143K31AgNK5pyVvW23gHnkBq2wh5aEk6g1s496M8ZMjxncCKZKgb5j\
ZoY5eSJMj2Vbyvi2hbmQnCuHBujZ2WXGTux1X2k9Krdtq

public version:

xpub661MyMwAqRbcFVF9ULcqlDsEa5WnCCugQAcgNd9iEMQ31tgH6u4DLQWoQayvtS\
VYFvXz2vPPpbXE1qpjoUFidhjFj82pVShWu9curWmb2zy

Other examples (some don't work on some machines):

<https://github.com/richardkiss/pycoin/blob/master/COMMAND-LINE-TOOLS.md>

Reference: Code with Other examples (some don't work):

<https://github.com/richardkiss/pycoin/blob/master/pycoin/scripts/ku.py>

Out of date blog entry (many don't work): <http://blog.richardkiss.com/?p=313>

Example Contd.

going 2 levels down in the tree:

```
genwallet -k
```

```
xprv9s21ZrQH143K3QTDL4LXw2F7HEK3wJUD2nW2nRk4stbPy6cq3jPPqjiChk  
VvvNKmPGJxWUtg6LnF5kejMRNNU3TGtRBeJgk33yuGBxrMPHi -s 0/1
```

better method with reading from a file (should contain the same xprv... as above):

```
genwallet -f ext-master-private-key.txt -s 0/1 > ext-private-key_0_1.txt
```

both should give the same result = xprv9ww7sMF...

Same with ku?

FAILS on many machines all versions of Python, 'Wallet' not defined

```
ku .py
```

```
xprv9s21ZrQH143K3QTDL4LXw2F7HEK3wJUD2nW2nRk4stbPy6cq3jPPqjiChk  
VvvNKmPGJxWUtg6LnF5kejMRNNU3TGtRBeJgk33yuGBxrMPHi -s 0/1
```

Contd. Private=>Public

Let `ext-master-private-key.txt` = contains=

```
xprv9s21ZrQH143K3QTDL4LXw2F7HEK3wJUD2nW2nRk4stbPy6cq3jPPqjiChkVvv  
  NKmPGJxWUtg6LnF5kejMRNNU3TGtRBeJgk33yuGBxrMPHi
```

Now produce the public extended key

```
genwallet -f ext-master-private-key.txt -s 0.pub
```

OR what is the difference?)

```
genwallet -f ext-master-private-key.txt -s 0p.pub
```

FAILS! Broken software!

Error = “public key has no private parts”

Should work, cf. <http://blog.richardkiss.com/?p=313>

Example Contd.

Call:

```
ku P:foo -j -s2/1/3 -n BTC
```

This produces private and public keys at branch /2/1/3!

```
ku P:foo -P -j -s2/1/3 -n BTC
```

This produces public sub-key at branch /2/1/3!

IMPORTANT: this a BAD and INSECURE method,
it exposes the secret seed in ps and shell's history!!!

(ps displays information about some active process)

Example Contd.

Or call:

ku

```
xpub661MyMwAqRbcFVF9ULcqLdsEa5WnCCugQAcgNd9i  
EMQ31tgH6u4DLQWoQayvtSVYFvXz2vPPpbXE1qpjoUFid  
hjFj82pVShWu9curWmb2zy -P -j -s2/1/3 -n BTC
```

FAILS for now... broken software...

NameError: name 'Wallet' is not defined

Again a BAD and INSECURE method,
it still exposes the public seed shell's history!!!

Example Contd.

Store the long string SECURELY, and SUBSTRINGS

Then call (to be tested):

```
ku -f master-public-key.txt -P -j -s2/1/3 -n BTC
```

This produces public sub-key at branch /2/1/3!

FAILS... NO WAY TO INPUT A FILE FOR NOW

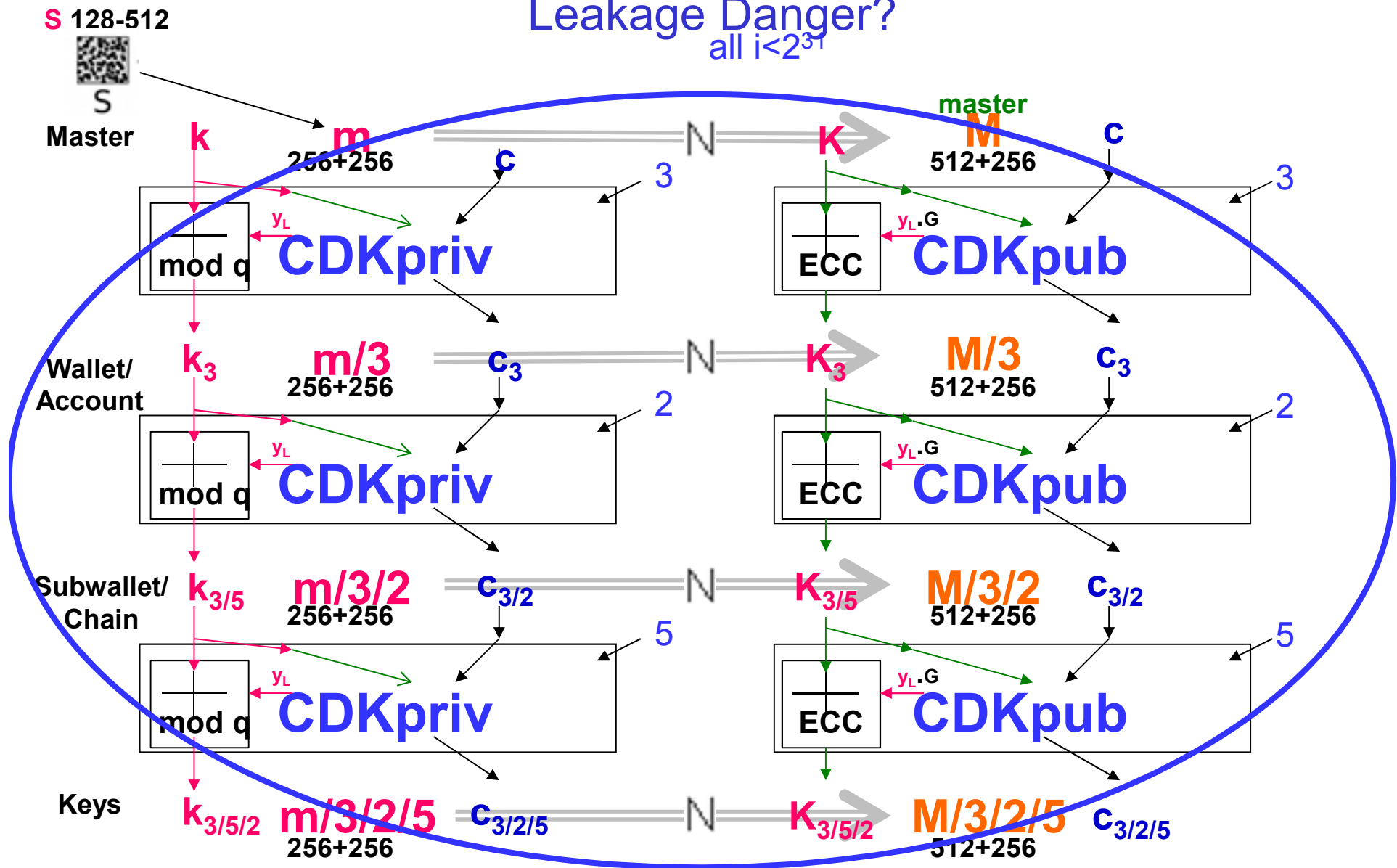
Better but still INSECURE method:

the file could be encrypted... Etc...

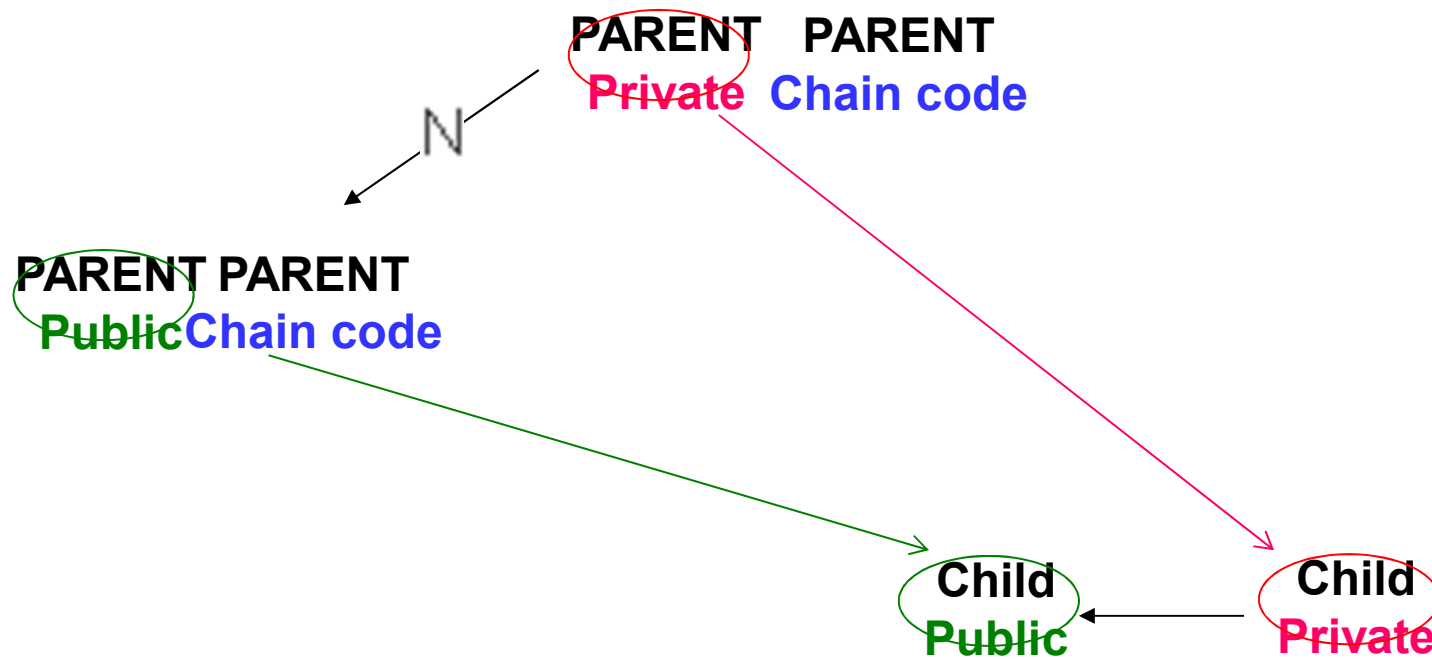
BIP032

Security

Leakage Danger?
all $i < 2^{31}$

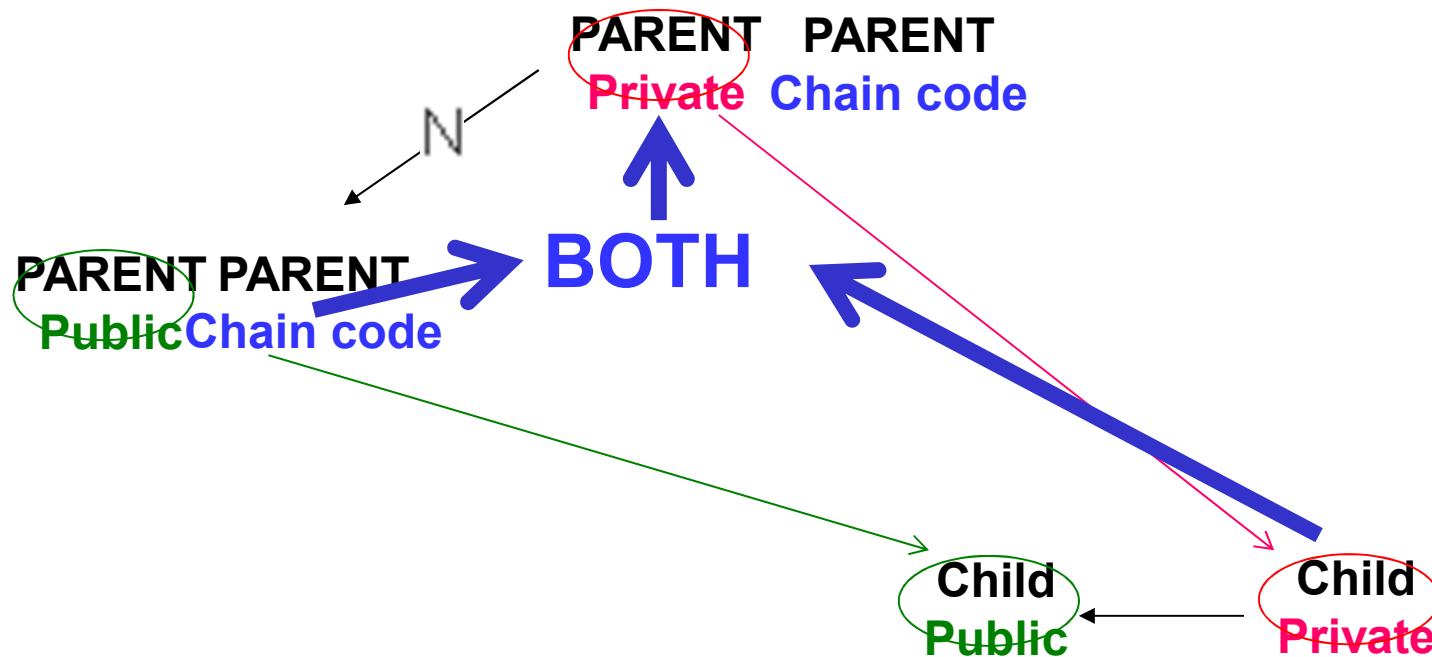


Leaks?



Leaks!

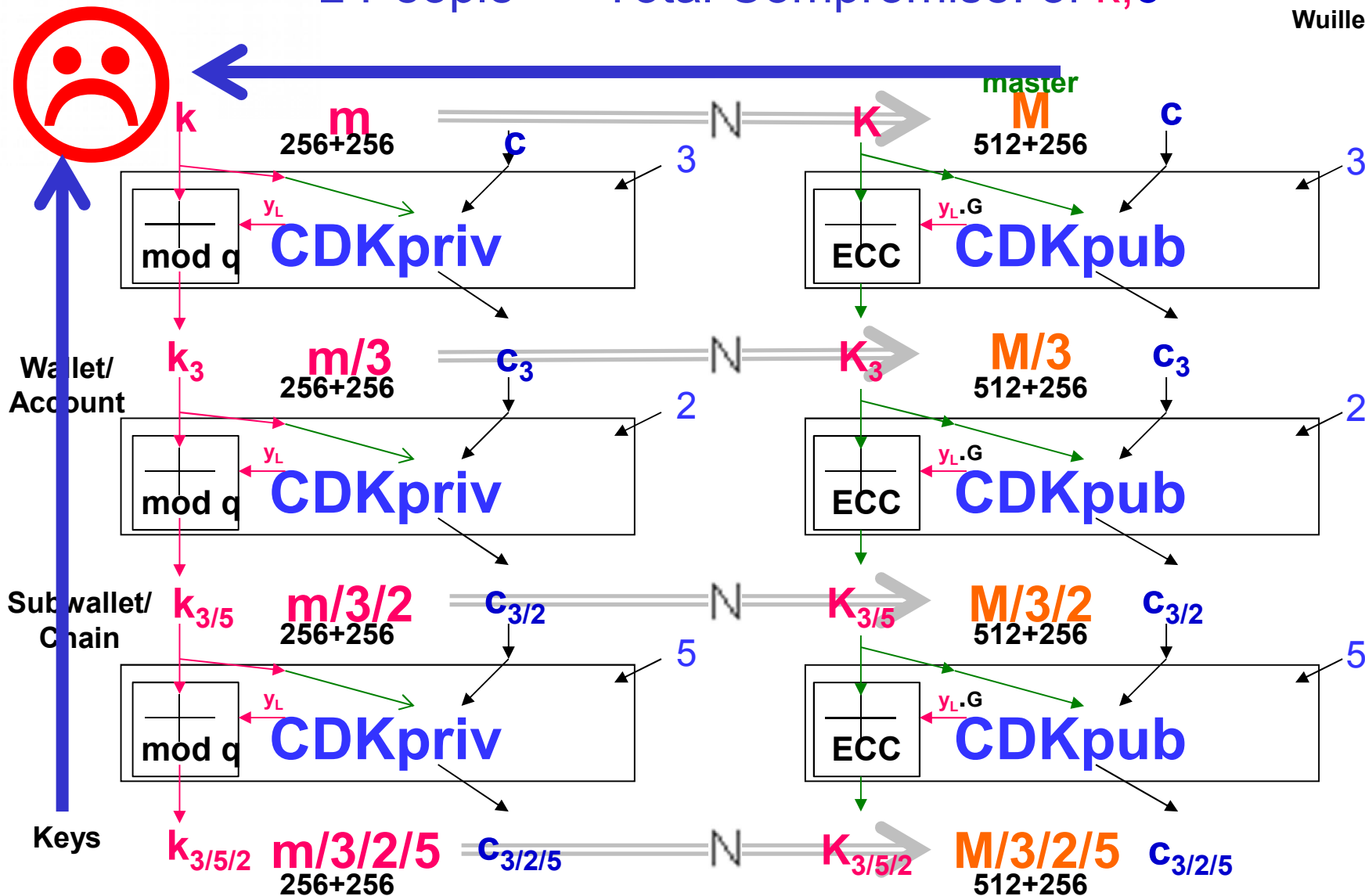
Jail breaking, 1 level up!



Q:
several levels up?

2 People => Total Compromise! of k, c

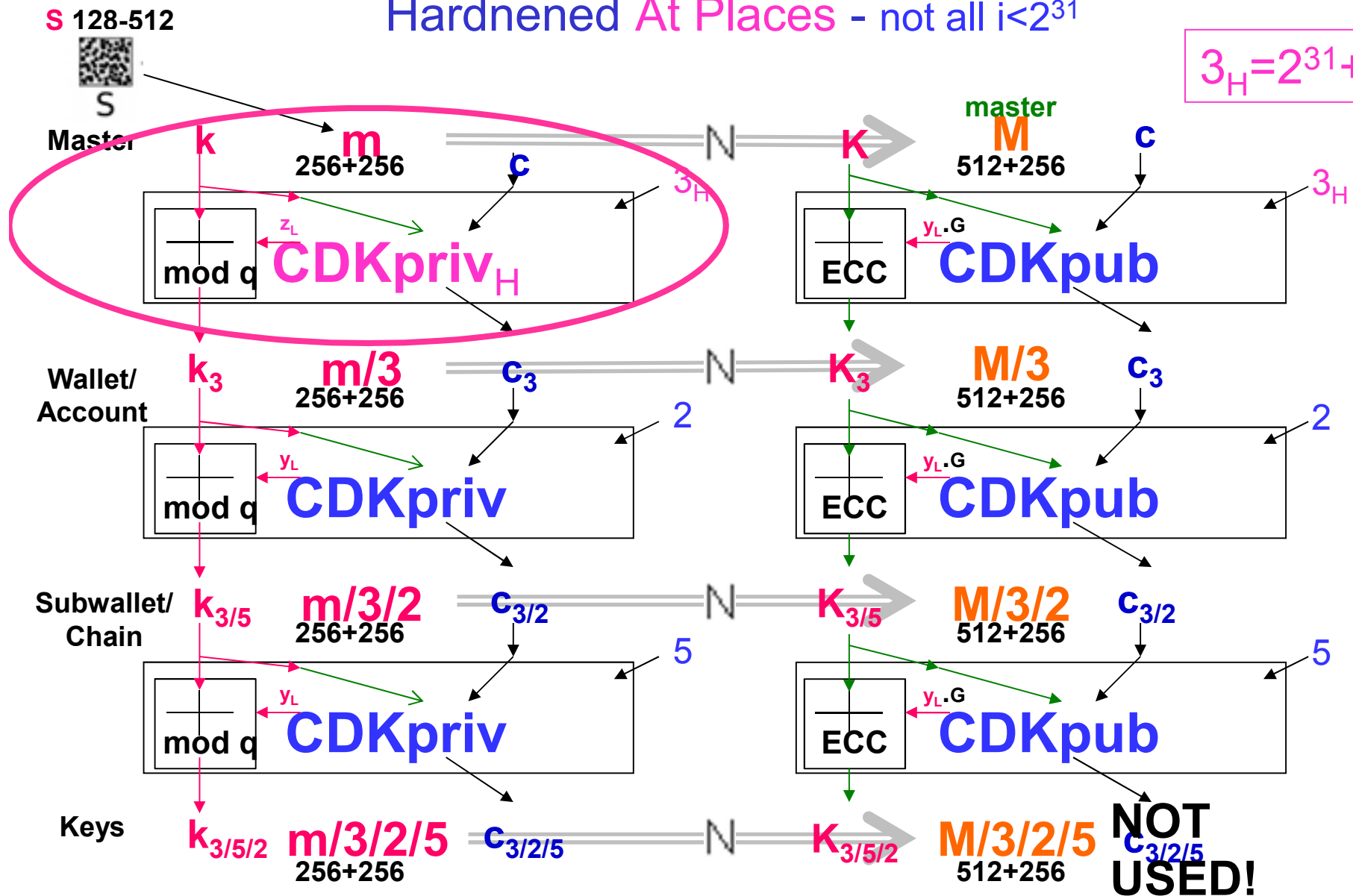
Pb. known to
Wuille etc.

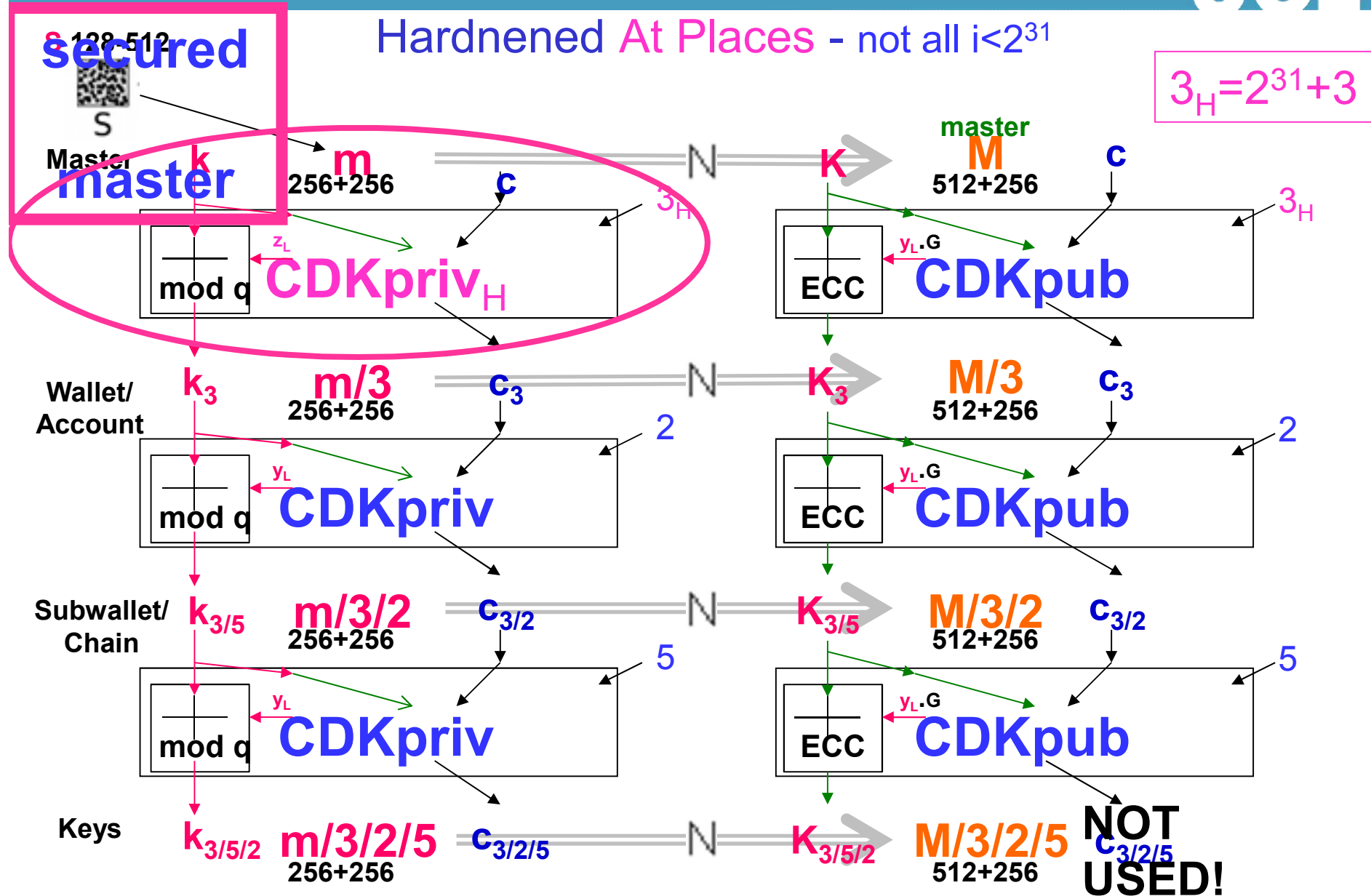


Hardening?

Hardened At Places - not all $i < 2^{31}$

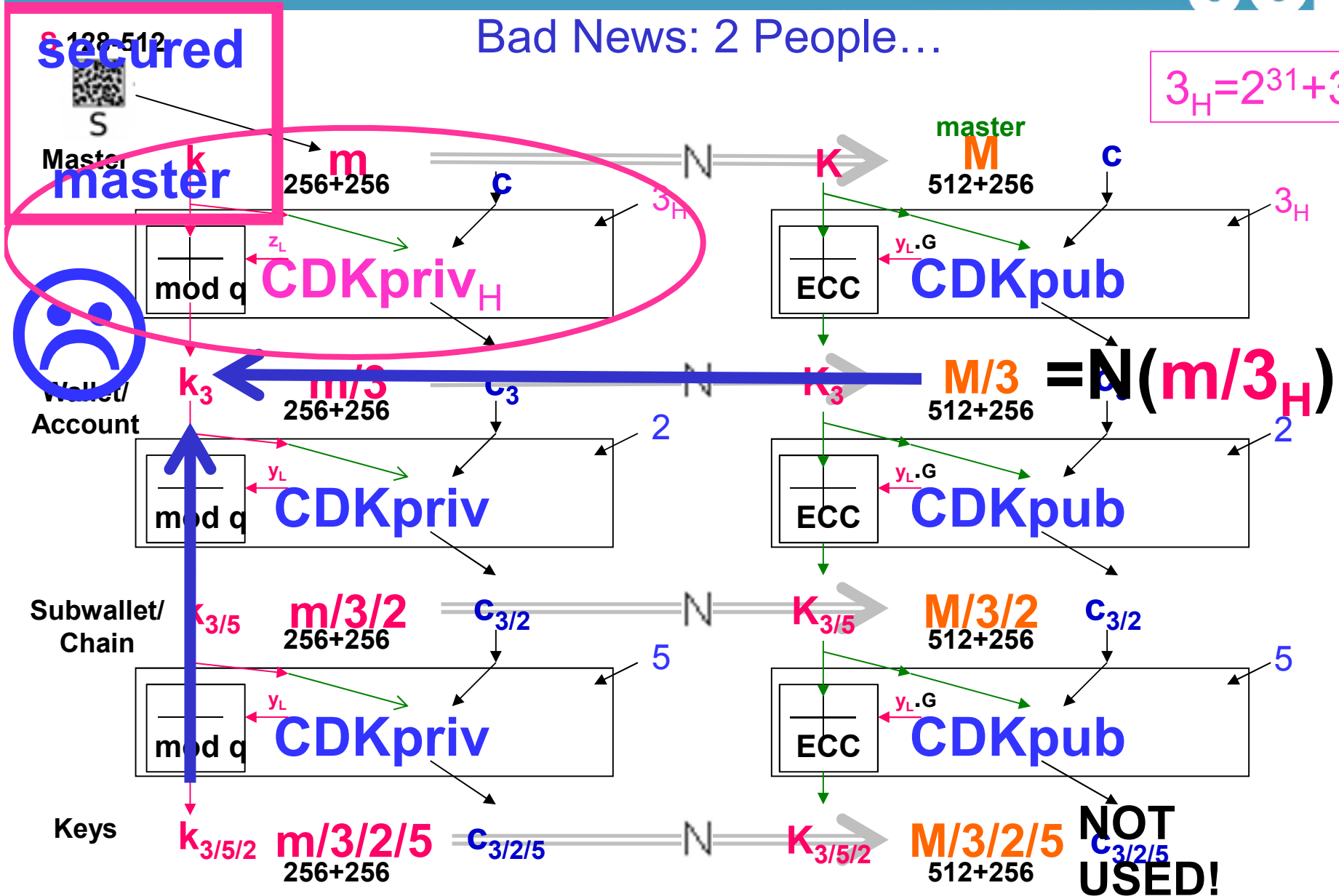
$$3_H = 2^{31} + 3$$





Bad News: 2 People...

$$3_H = 2^{31} + 3$$



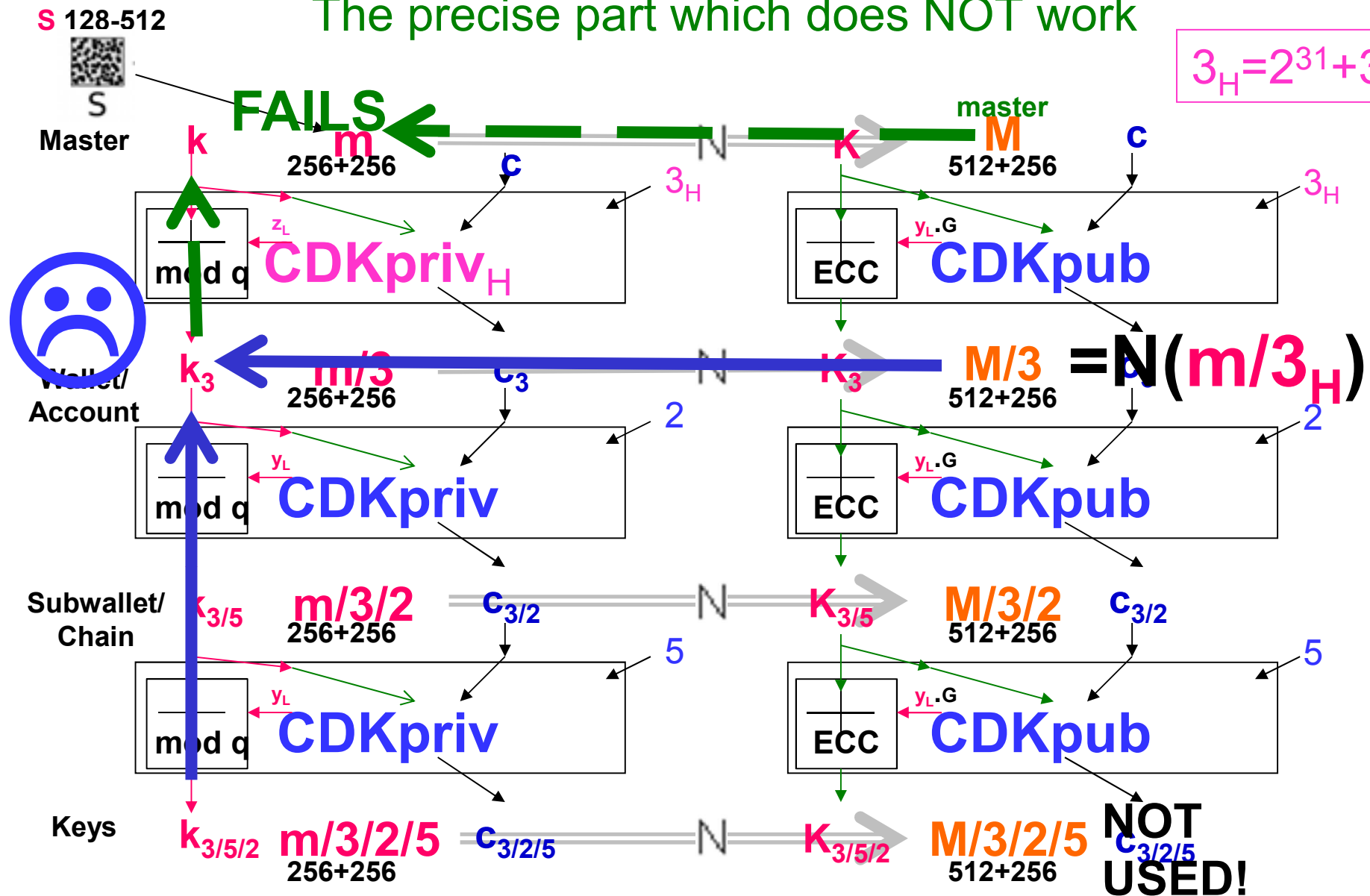


$$3_H = 2^{31} + 3$$



The precise part which does NOT work

$$3_H = 2^{31} + 3$$



Lose Magic Audit Capability?

We don't. Or it can be restored [new!].

Mix of key derivation and database approach is POSSIBLE!

*Recall: Sub-domains

Longer definition was given before

Definition: We call **a security sub-domain** a set of all the nodes below one place on any side [...]

e.g. a domain under $m/1$ (left)

or under $M/0/1 = N(m/0/1)$ (right).

Hardened =>

Hardened => Only certain sub-domains can be audited with 1 key.

Definition: We call **an auditable sub-domain** each domain under the **last** hardened index i , for example $N(m/a_H/b/c_H)$ is the corresponding **extended public sub-domain key**.

(always on the right hand side in our earlier diagrams)

Note: it is assuming that all indexes below c_H are NOT hardened, for example $d < 2^{31}$.

Theorem: The auditor just needs to know the **extended public sub-domain key** root for EACH **auditable sub-domain branch**. For example both $N(m/3_H/any/1_H)$ and $N(m/2_H)$.

again assuming that all indexes below these branches are NOT hardened.

Does it REPAIR Type 2 Wallets?

In one way it does.

However, in such systems there are still relatively valuable keys: a hacker who steals 2 keys from 2 different places **CAN** have access to lots of accounts with maybe tens of millions of dollars.

A **sensible policy** would be to limit the total amount stored at each **auditable sub-domain branch** as defined above.

=> The more we do it, the closer we get to the super-paranoid solution: give the auditors ALL public keys (larger database, harder to manage, update, add users, fresh return addresses etc...).

The Converse Result

Theorem 2: In order to achieve total compromise of all keys, the hacker needs to gather one private key from EACH **auditable sub-domain branch** and one top-level audit key from each branch.

Attacks with **INDIVIDUAL KEYS** and **Without Auditors?**



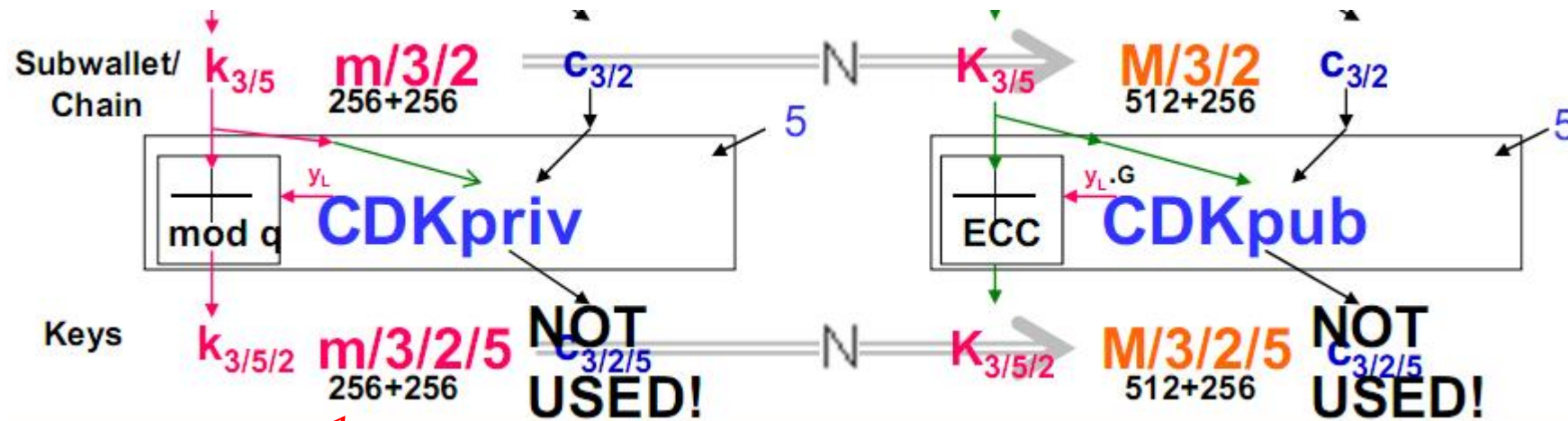
More Attacks?

Other attacks on HD Wallets?

- These would NOT involve auditors.
- They would one or several users with their **legitimate** private and public keys.
- Could be much harder to detect:
 - Powerful auditors can be suspected,
 - HERE only a conspiring **ORDINARY** users
 - Could be just running the **same application**,
 - which also implements the attack...

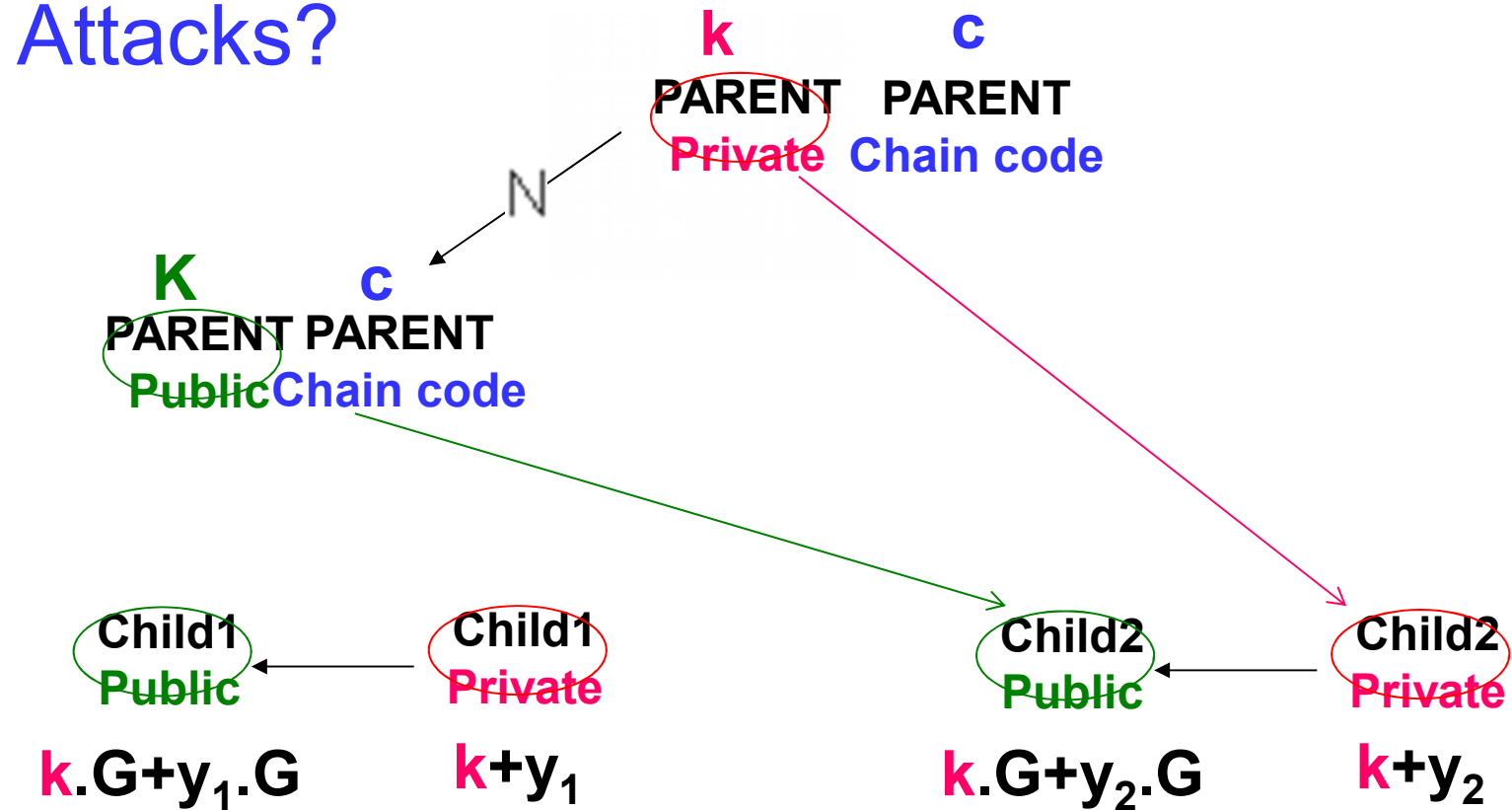
More Attacks?

For now assume 2 users, SAME parent.



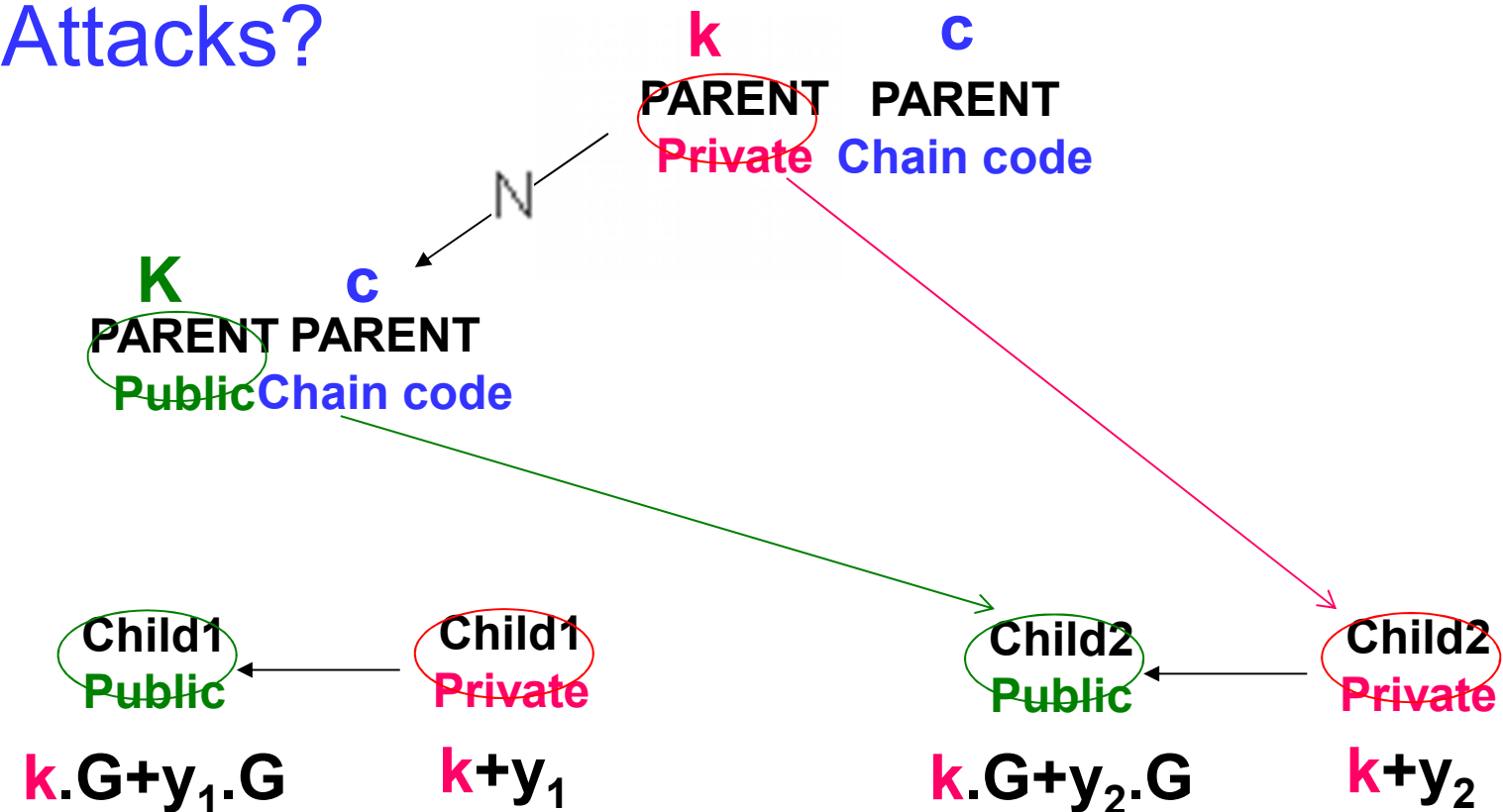
Say $m/3/2/5$ and $m/3/2/6$.

Attacks?



Claim: $y_1 - y_2$ may be revealed ≤ 2 private keys

Attacks?



Claim: $y_1 - y_2$ may be revealed ≤ 2 private keys

Theorem 1. None of k, K, c, y_L are EVER revealed!
Not even for more users.

Further Results: [to be proven later...]

Theorem 2. If ECDL is hard + extra assumptions? =>

- Same parent, 2 children.
 - any of 4 child keys=> master private **k** is never revealed
 - proof: this would solve ECDL?
 - any of 4 child keys=> master public **K** is never revealed
 - proof??
 - 2 public 1 private => cannot compute other private.
 - proof??
 - 2 private => can compute 2 public obviously
- Same parent, more children: same results
- Arbitrary HD Wallet with any number of levels, any mix of low level and high level keys: cannot derive more than what follows from 2 rules: auditor jail-breaking + down the graph.

Other Attacks?



Other Attacks?

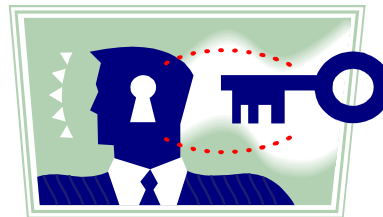
1. Weak keys:

- Weak keys: imagine that the NSA can break 0.00001% of all secp256k1 keys.

=>HD Wallets audit functionality
REVEALS ALL PUBLIC KEYS.

- if NSA can break just SOME of secp256k1 keys we are toast.
 - they will be able to select themselves which keys are “breakable” and crack them: steal bitcoins.
- IN CONTRAST if bitcoin is used in a certain careful/secure way, public keys are NEVER revealed in advance...

Practical Problems

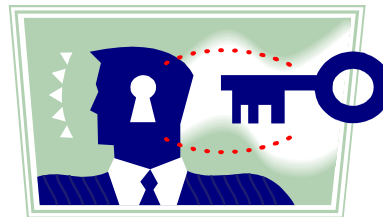


Did We Receive Money?

If we give sb. some high-level extended PUBLIC key (K+chain), we CANNOT know if money are received in sub-wallets

- unless the number of accounts used is small (like small integers).
- with 3 levels already 2^{96} possible receiving accounts!

Secure Solutions = Leakage Resistance



Motivation

Most system seen so far are FRAGILE:

- ⇒ One key can compromise all
- ⇒ The more keys you have the more likely is that leakage of "one which compromises all" WILL occur
 - ⇒ SCA,
 - ⇒ kleptography,
 - ⇒ bugs inside CPUs meant to help keys to leak...
 - ⇒ etc..

We need:

- ⇒ Leakage resistance.

Solution

Gus Gutoski and Douglas Stebila:

Hierarchical deterministic Bitcoin wallets that tolerate key leakage,

in Financial Cryptography 2015,

preprint at <https://eprint.iacr.org/2014/998>

"Our HD wallet can tolerate the leakage of up to m private keys with a master public key size of $O(m)$ ".