black hat ASIA 2024

APRIL 18-19, 2024

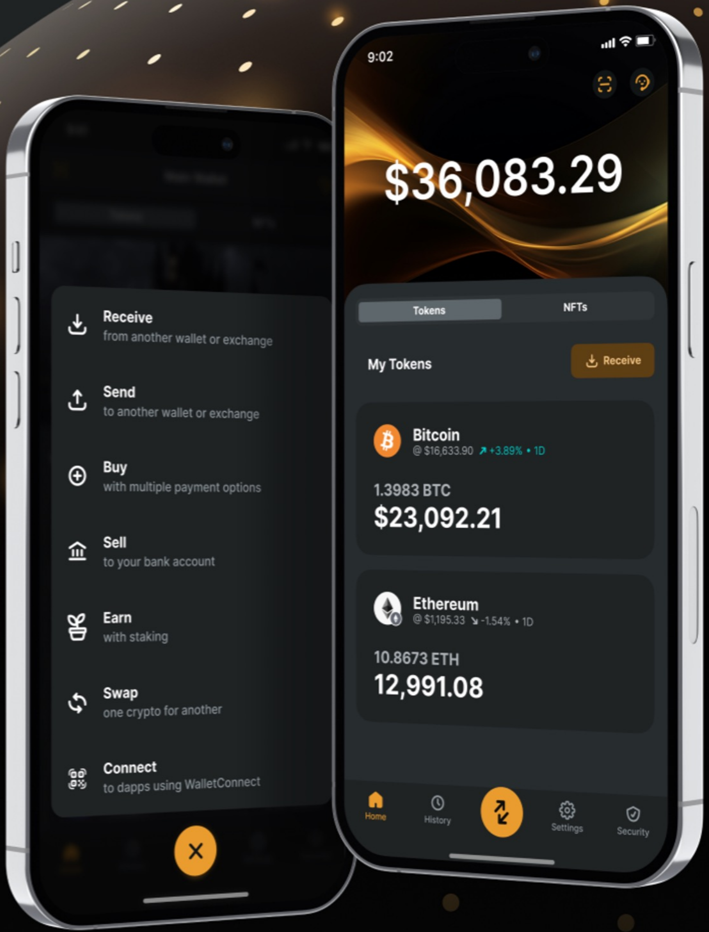
BRIEFINGS

Bad Randomness: Protecting Against Cryptography's Perfect Crime

Tal Be’ery, CTO & Co-Founder Zengo

👋 Hi, I'm Tal Be'ery

- Co-Founder, CTO @ ZenGo
- 20+ years cyber security
- 9th time BH Speaker
- 1st time BHASIA speaker!
- @talbeerysec



Agenda

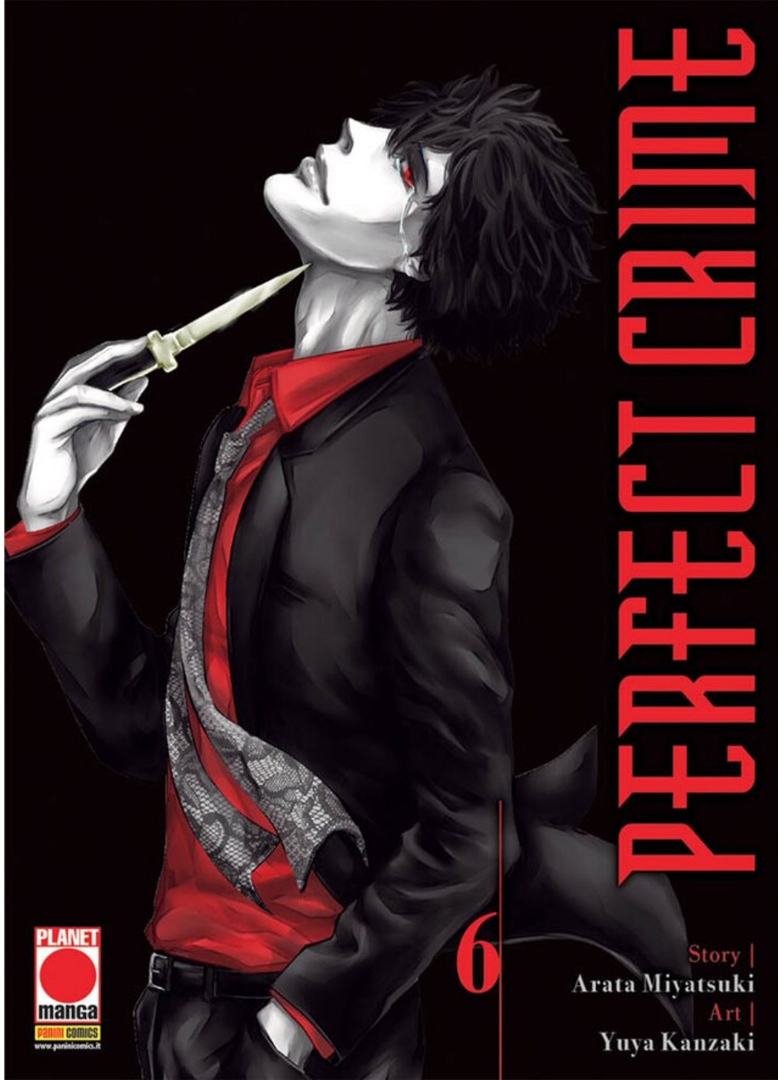
- The Perfect Crime: Why bad randomness is crypto's perfect crime?
- True Crime(s)
  - Bad private key: Bitcoin, gone in milliseconds
  - Bad Nonce: Ethereum, gone in milliseconds
  - Bad DH parameters: TLS malware, even more powerful than previously known
- Solutions
  - Avoiding single point of failure with MPC

The perfect crime

Randomness in cryptography

The perfect crime

- Lethal
- Undetectable



“

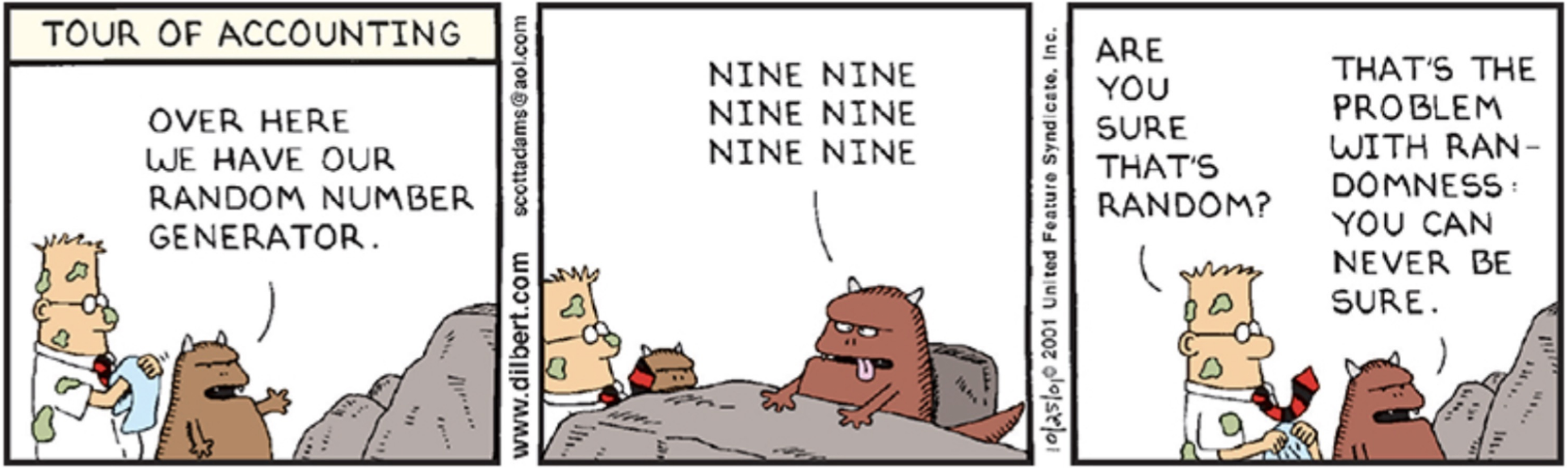
Randomness in cryptography is like the air we breathe. You can’t do anything without it,

- Prof. Yevgeniy Dodis https://cs.nyu.edu/~dodis/courant-article.pdf

Randomness is vital

- Kerckhoffs' principle: the security of a cryptographic system should be based on the secrecy of the cryptographic key
- Keys values should be unguessable
  - ➔ created in random
- But also other crypto items, e.g. Nonces, IVs
- Randomness is vital ➔ Lack thereof is lethal!

Bad randomness is undetectable



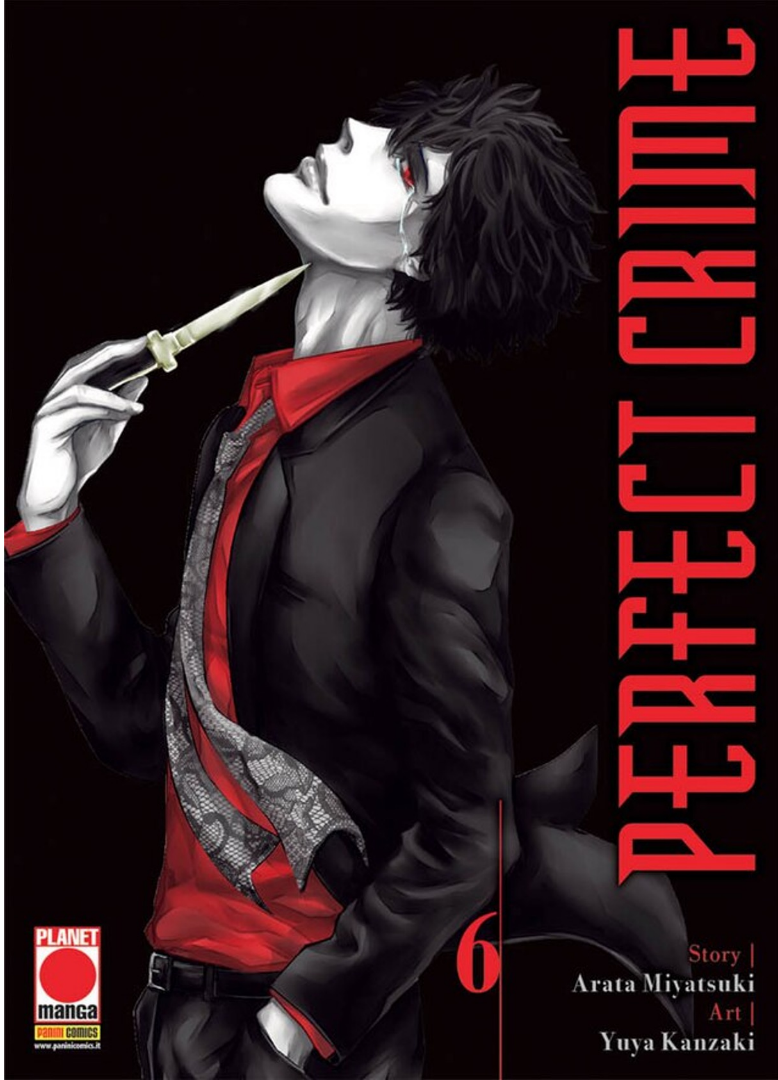
Bad randomness is undetectable

- There are no random numbers, only numbers created by a random process
- In most cases, you cannot inspect a number and decide if it is random or not
- In most cases, the values of these random numbers are not stored as they are too secret → not available for a statistical forensic analysis

Crypto’s perfect crime

Bad randomness is crypto’s perfect crime

- **Lethal**
- **Undetectable**



True crime, true detective

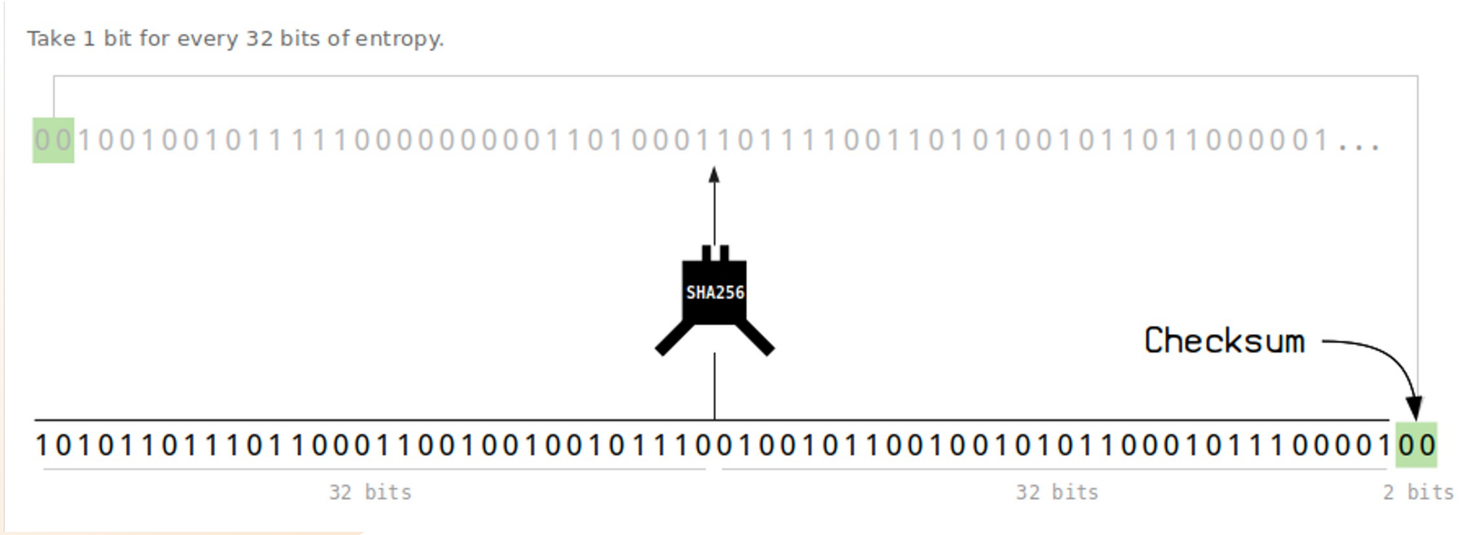
Bad Randomness in the wild

True detective

Season 1: Bitcoin's dark forest

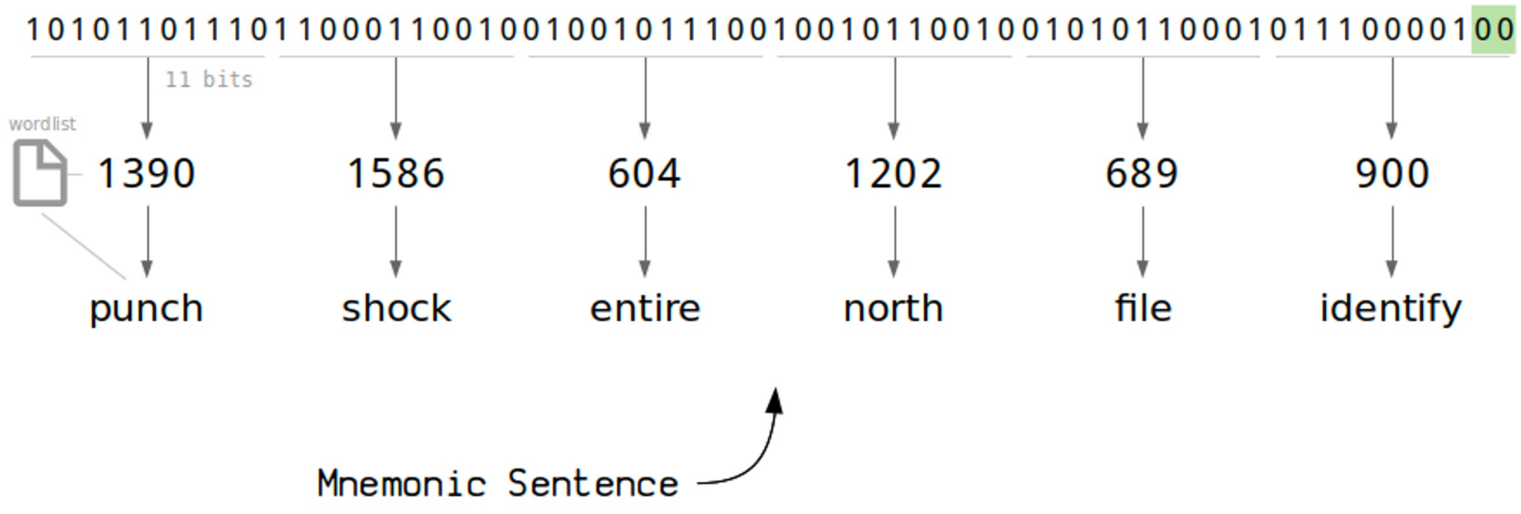
From random to Bitcoin address: step 1

- Generate a random 128 bit number
- Add 1 bit of checksum for each 32 bit (33 is divisible by 11)



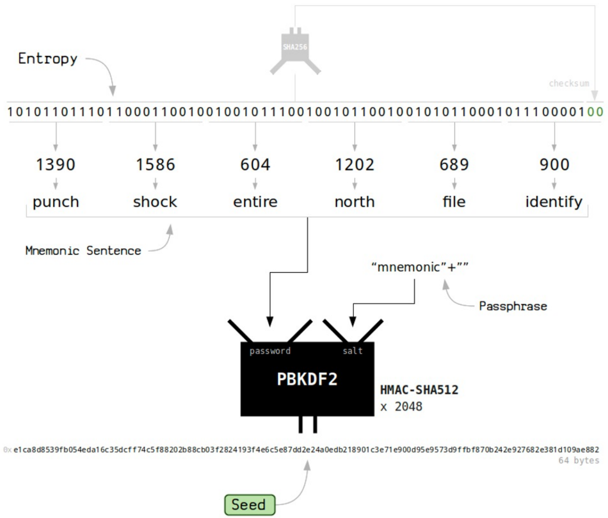
From random to Bitcoin address: step 2

- Assign for each 11 bit group a word from [BIP-39](BIP-39) to get the seed phrase



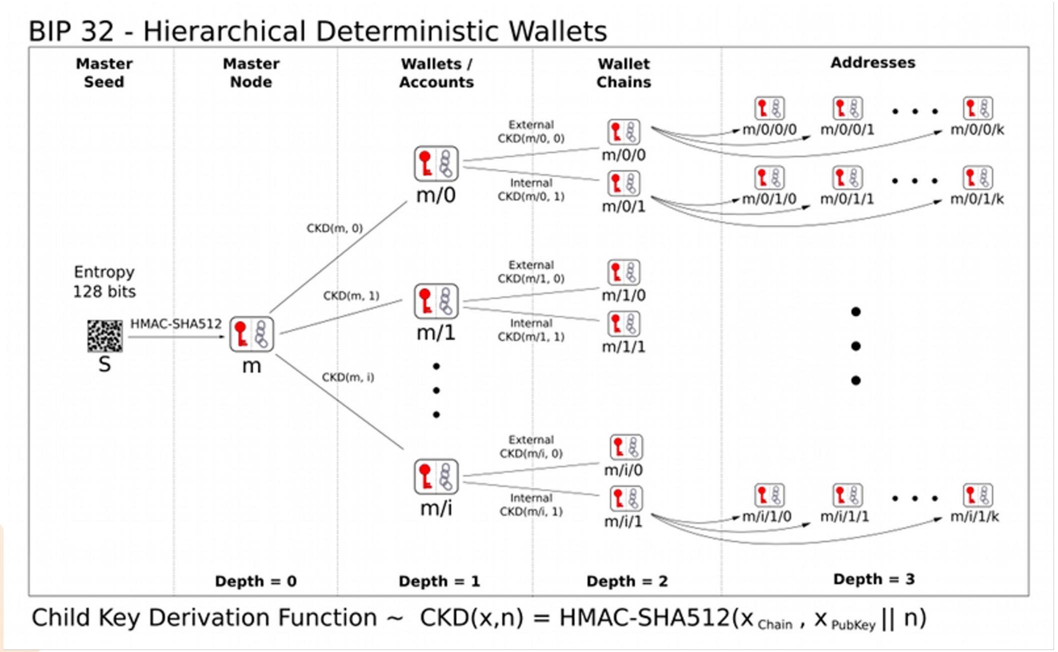
From random to Bitcoin address: step 3

- **Key Derivation Function: PBKDF2: 2048 HMAC-SHA512**
- **Adding performance "penalty" to make bruteforce harder**



From random to Bitcoin address: step 4

- Derive addresses



Randomness in crypto addresses

- Getting an address might be a complex process
- But it all starts with a random number
- If this number is guessable, all funds are gone!

Bad randomness can cost Billions

If you created a bitcoin wallet before 2016, your money may be at risk

A company that helps recover cryptocurrency discovered a software flaw putting as much as $1 billion at risk from hackers. Now it’s going public in hopes people will move their money before they get robbed.

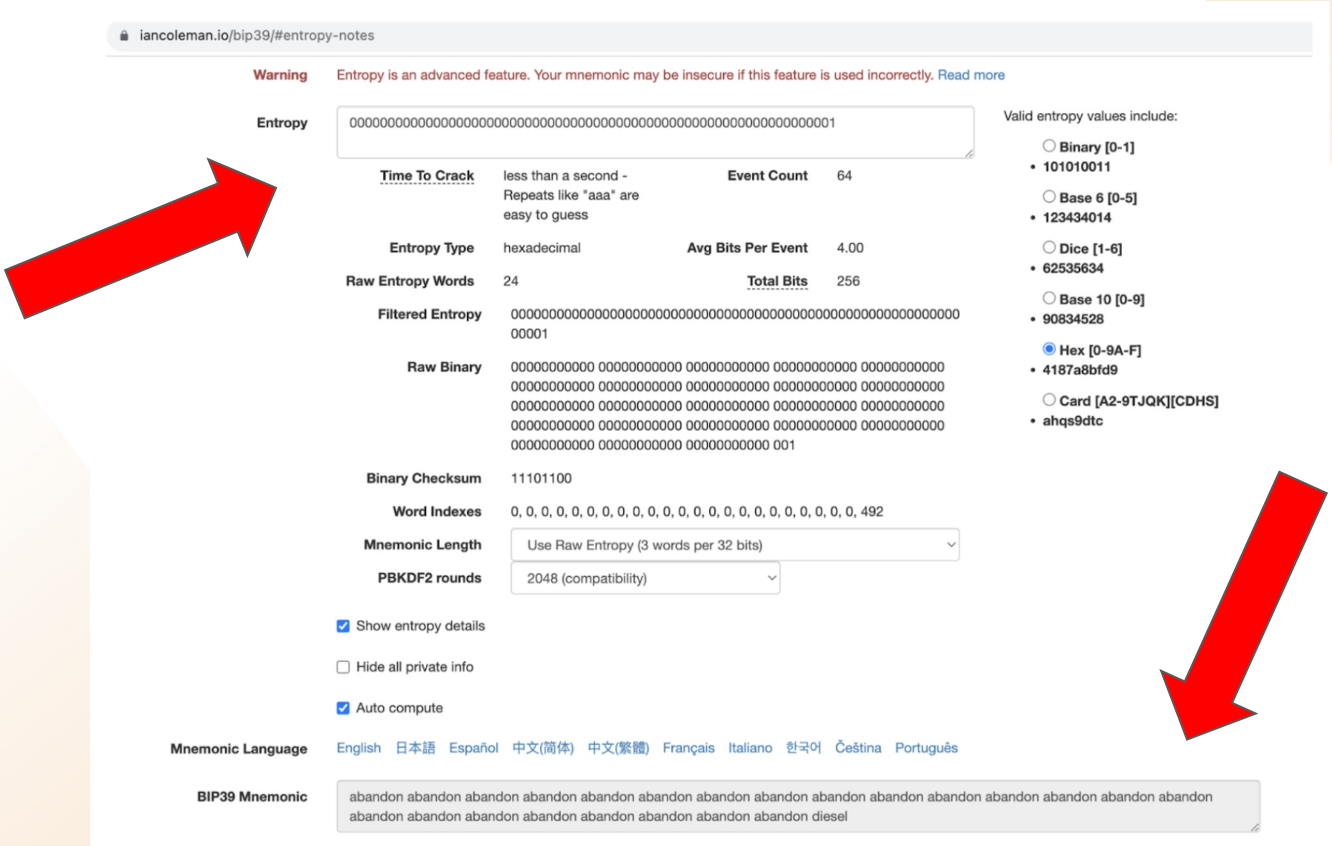
By Joseph Menn

Updated November 14, 2023 at 1:30 p.m. EST | Published November 14, 2023 at 6:00 a.m. EST

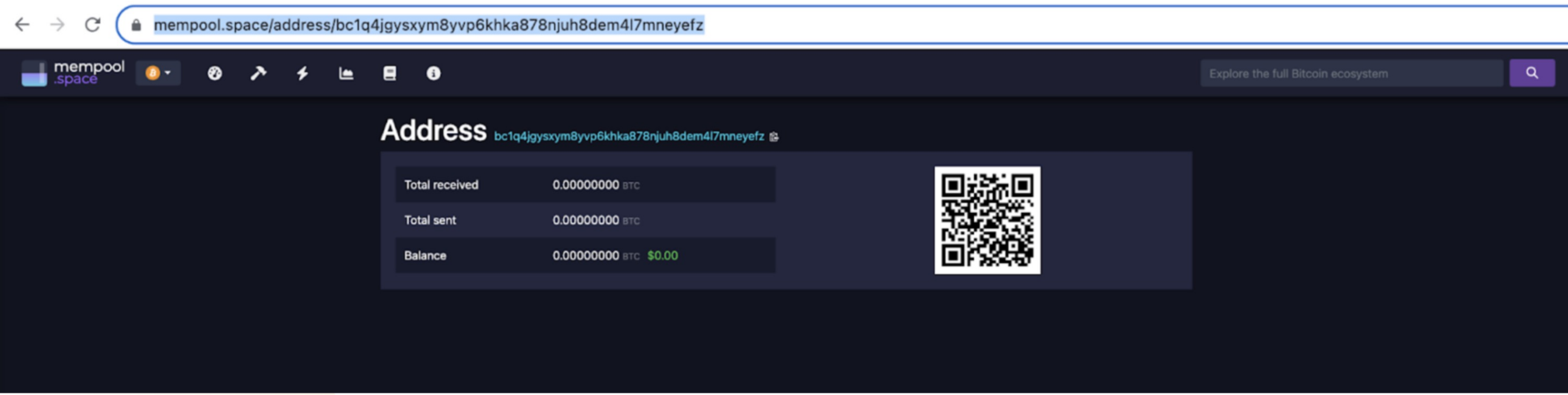
https://www.washingtonpost.com/technology/2023/11/14/bitcoin-wallet-passcode-flaw/

POC!

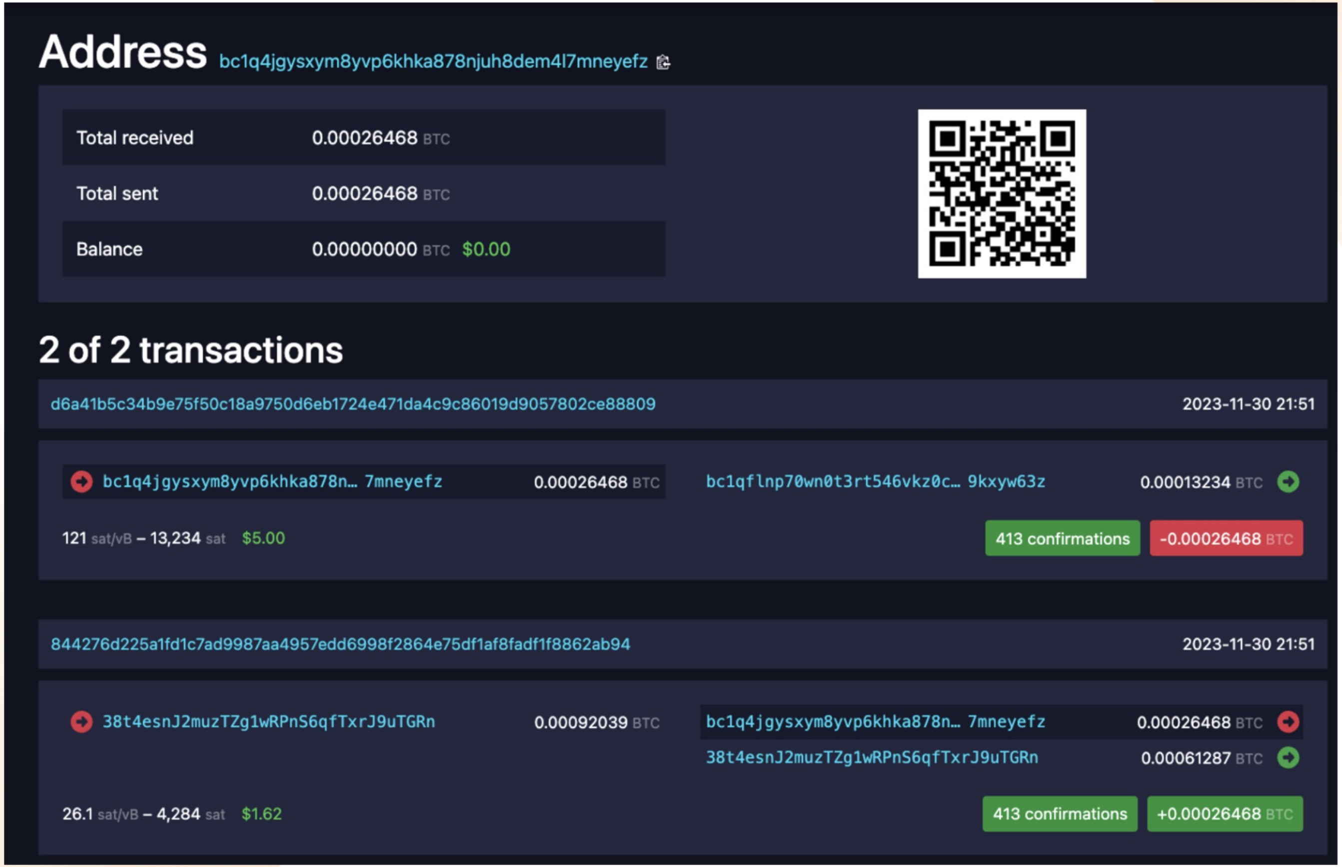
Step 1: bad randomness Bitcoin Key



Step 2: Address is pristine



Step 3: Send money.. It's gone!



Conclusions

- Bad randomness attackers are real
- Bots are lurking for transactions to bad randomness addresses and taking them away in real time
- Further reading
  - → https://zengo.com/how-keys-are-made/
  - → https://zengo.com/bitcoin-is-a-dark-forest-too/

True detective

Season 2: Ethereum’s dark forest

ECDSA nonce

- ECDSA signatures are used in many security related protocols
  - → Authentication
  - → Cryptocurrency
- require a nonce that should be secret → let's make it random
- However if nonce is somewhat predictable..
- LadderLeak: Breaking ECDSA with Less than One Bit of Nonce Leakage (BH EU 2020)

Nonce reuse dark forest in the wild

@bertcmiller ⚡🤖
@bertcmiller

Last week a monster in Ethereum's dark forest revealed themselves to me.

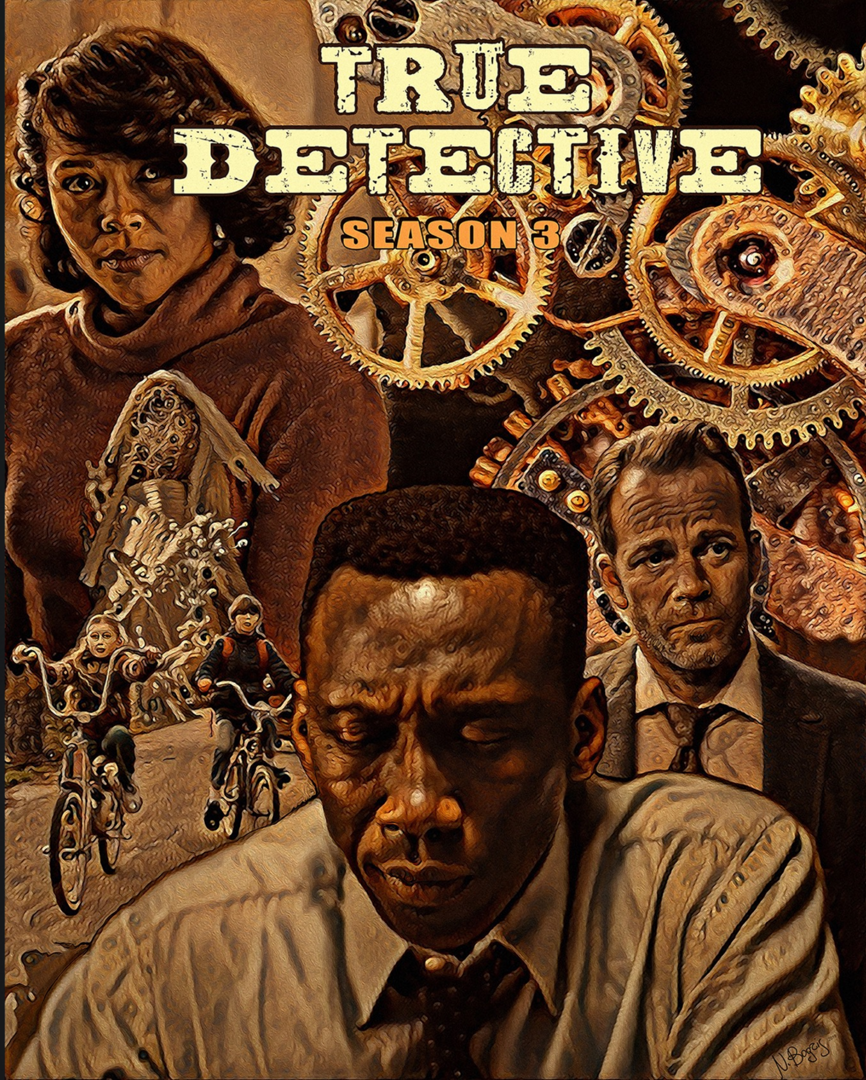
This blog post tells the story of that encounter:
bertcmiller.com/2021/12/28/gli...

5:03 PM · Dec 28, 2021

https://twitter.com/bertcmiller/status/1475844939816833032

True detective

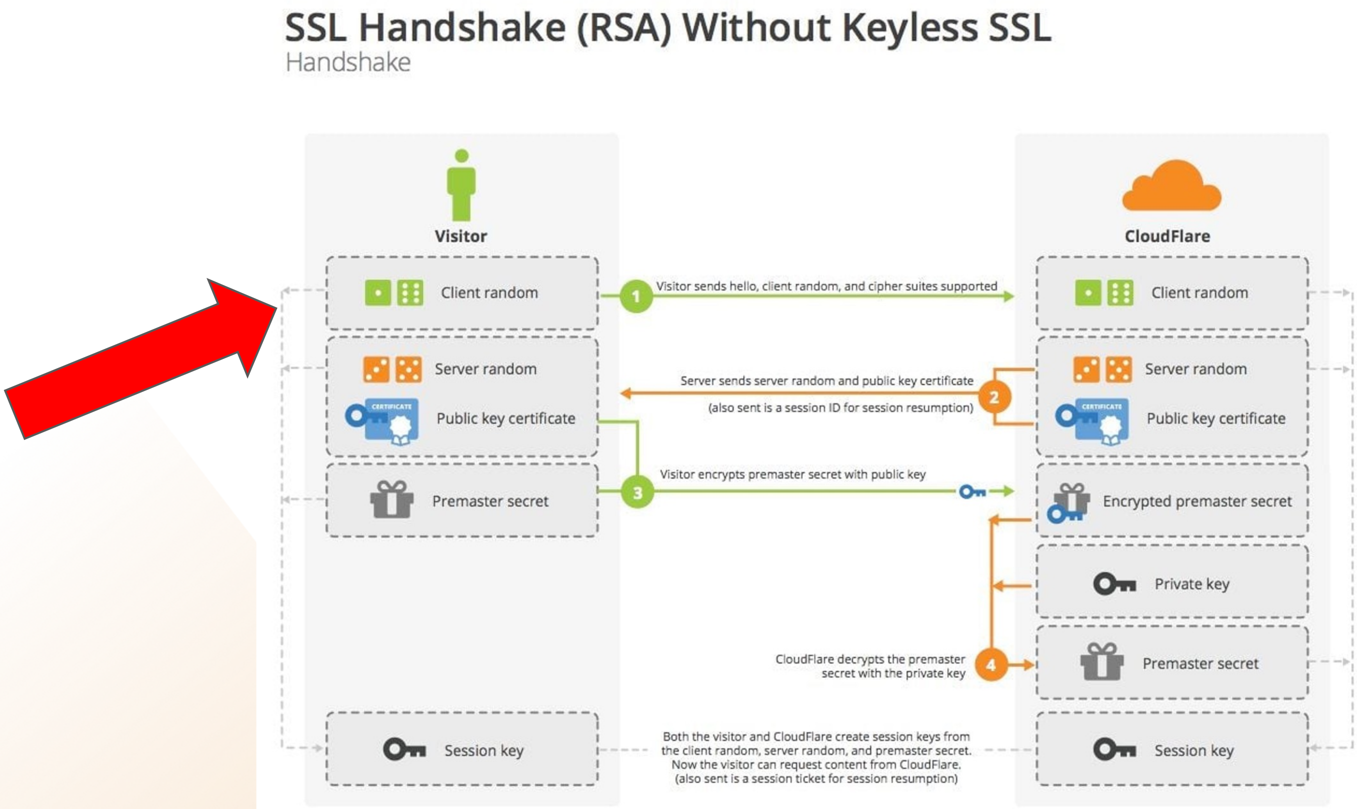
Season 3: The TLS malware



The Reductor Malware

- Identified by Kaspersky in 2019
  - → https://securelist.com/compfun-successor-reductor/93633/
  - → Attributed to Turla APT group
- Malware:
  - → patches the PRNG
  - → injects CA TLS Certs

The TLS Handshake



https://blog.cloudflare.com/keyless-ssl-the-nitty-gritty-technical-details

Patching the PRNG: The Code POV

PRNG functions

"nss3.dll"	PK11_GenerateRandom()	Call original PRNG function and generate initial XOR key from its result. Change PRNG result: set seventh byte to 1, then save 0x45F2837D, hwid and cert hashes. Encrypt the result and return it instead of the original PRN. It will affect calls to ssl3_SendClientHello() -> ssl3_GetNewRandom(ss->ssl3.hs.client_random);
"advapi32.dll"	CryptGenRandom()	Spoof these system PRNG function results in similar way with some minor changes;
"bcrypt.dll"	BCryptGenRandom()	
"chrome.dll"	PRNG function	Find PRNG function by its binary code template and patch it like all the aforementioned.

Patching the PRNG: The network POV

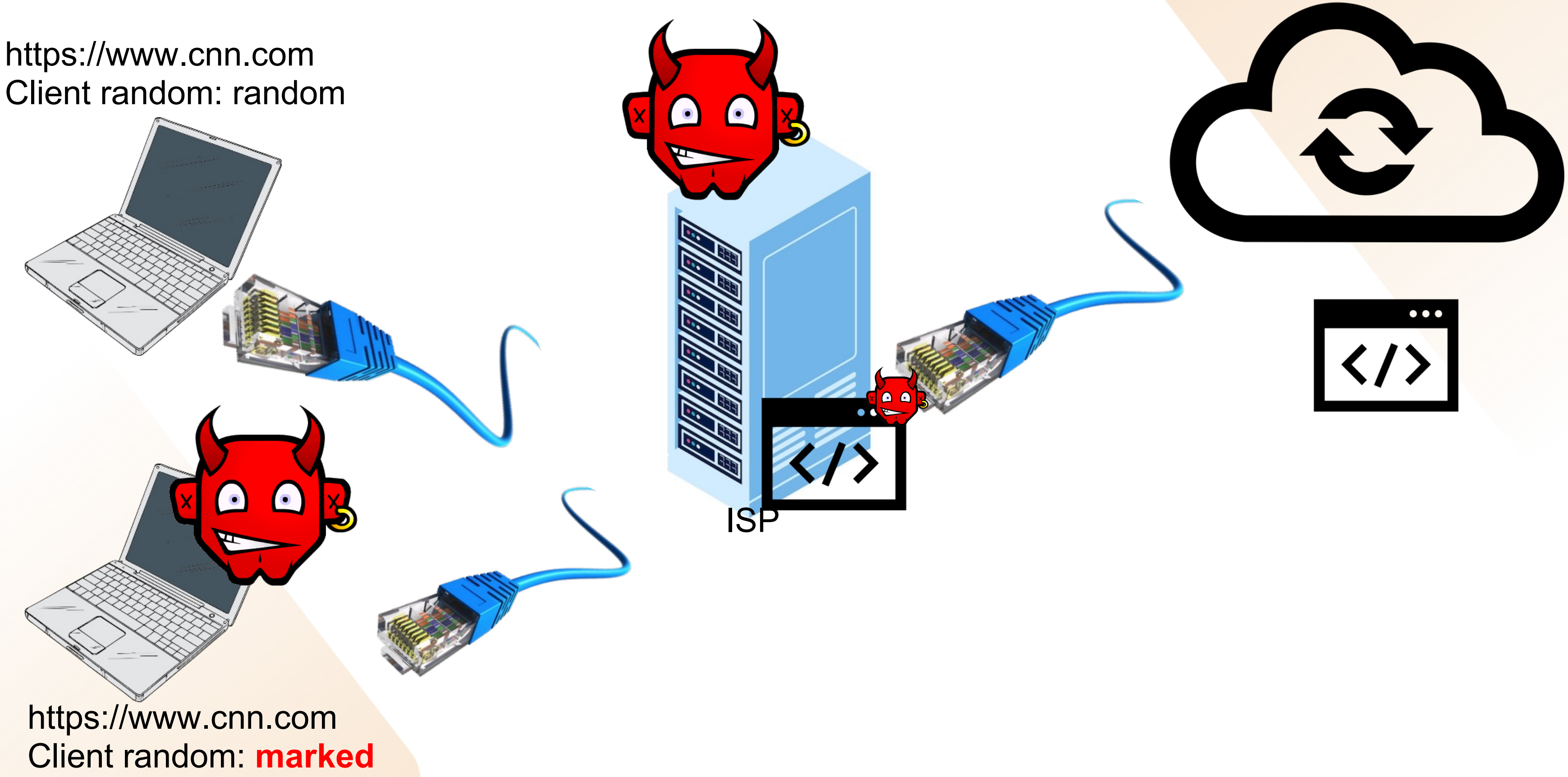
```
Handshake Protocol: Client Hello
  Handshake Type: Client Hello (1)
  Length: 126
  Version: TLS 1.2 (0x0303)
  Random: 64d34a0100000000000000000000000000000000000000000000000000000000
    GMT Unix Time: Aug  9, 2023 11:10:41.000000000 IDT
    Random Bytes: 00000000000000000000000000000000000000000000000000000000
```

Cyber paleontology

- Reductor malware:
  - ➔ patches the PRNG
  - ➔ injects CA TLS Certs
- Reductor malware must be working with a server MITM

https://www.kaspersky.com/blog/cyberpaleontology-managed-protection/24118/

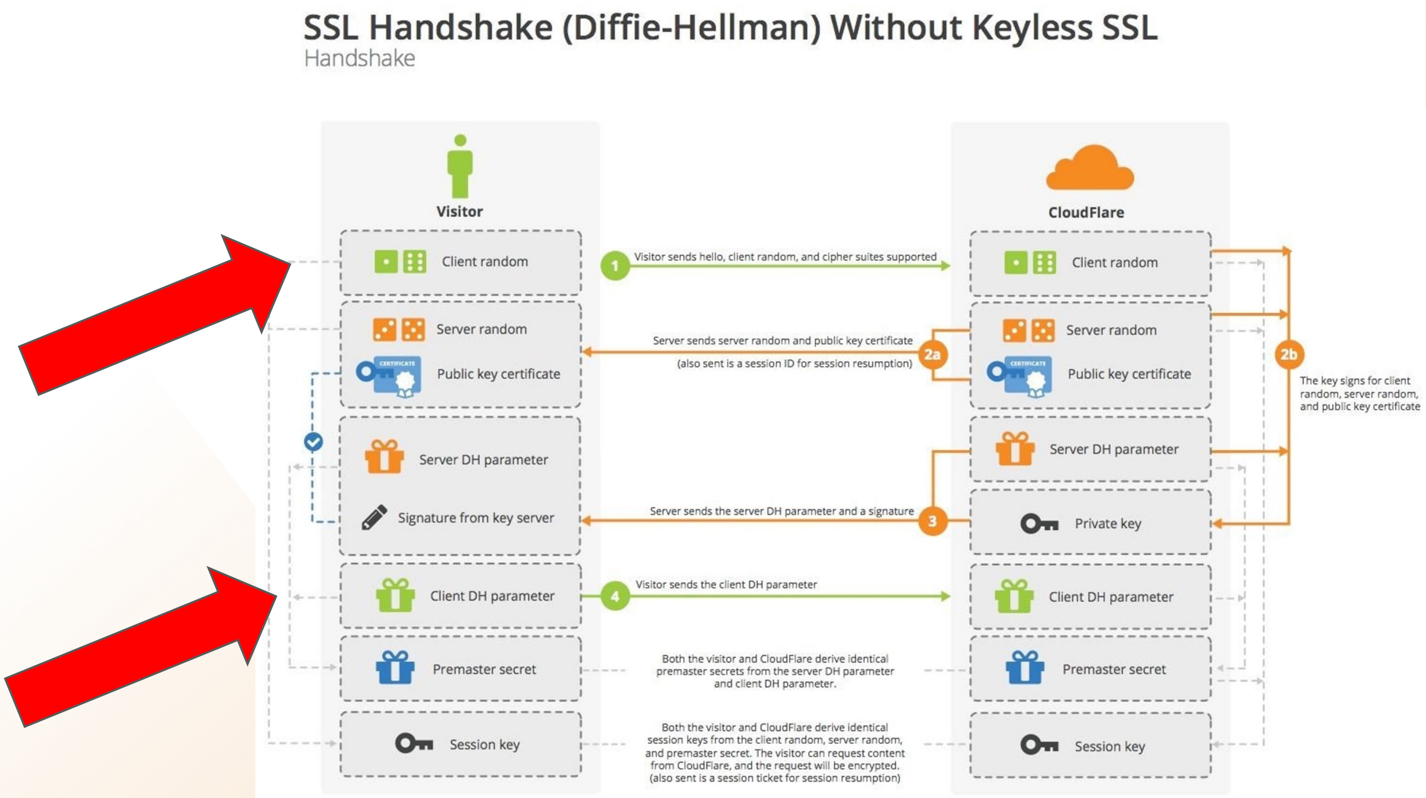
The Reductor MITM: Active MITM



Some observations

- Monsters (Bad randomness attackers) are real!
- Although attackers can use their malware, they prefer to fiddle with network traffic
- Why?
  - → Does not really matter
  - → More stealthy

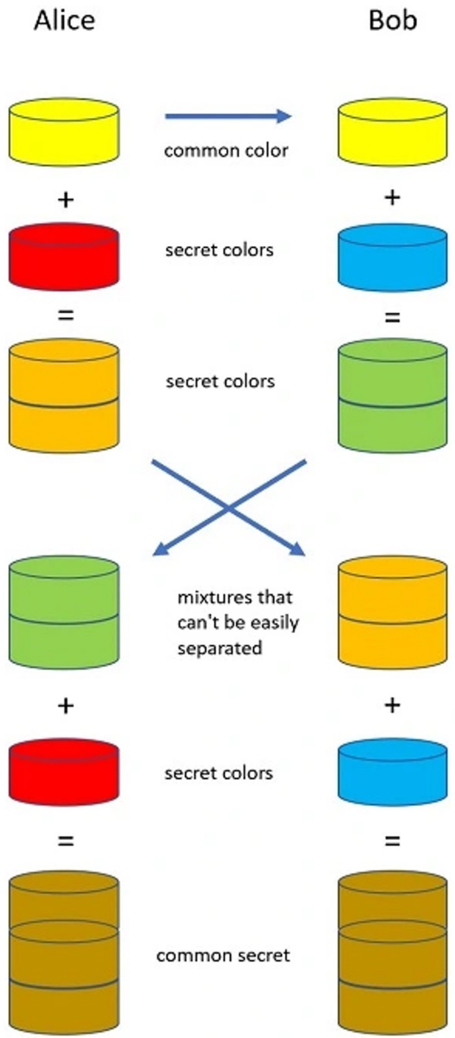
The TLS Handshake with EDH



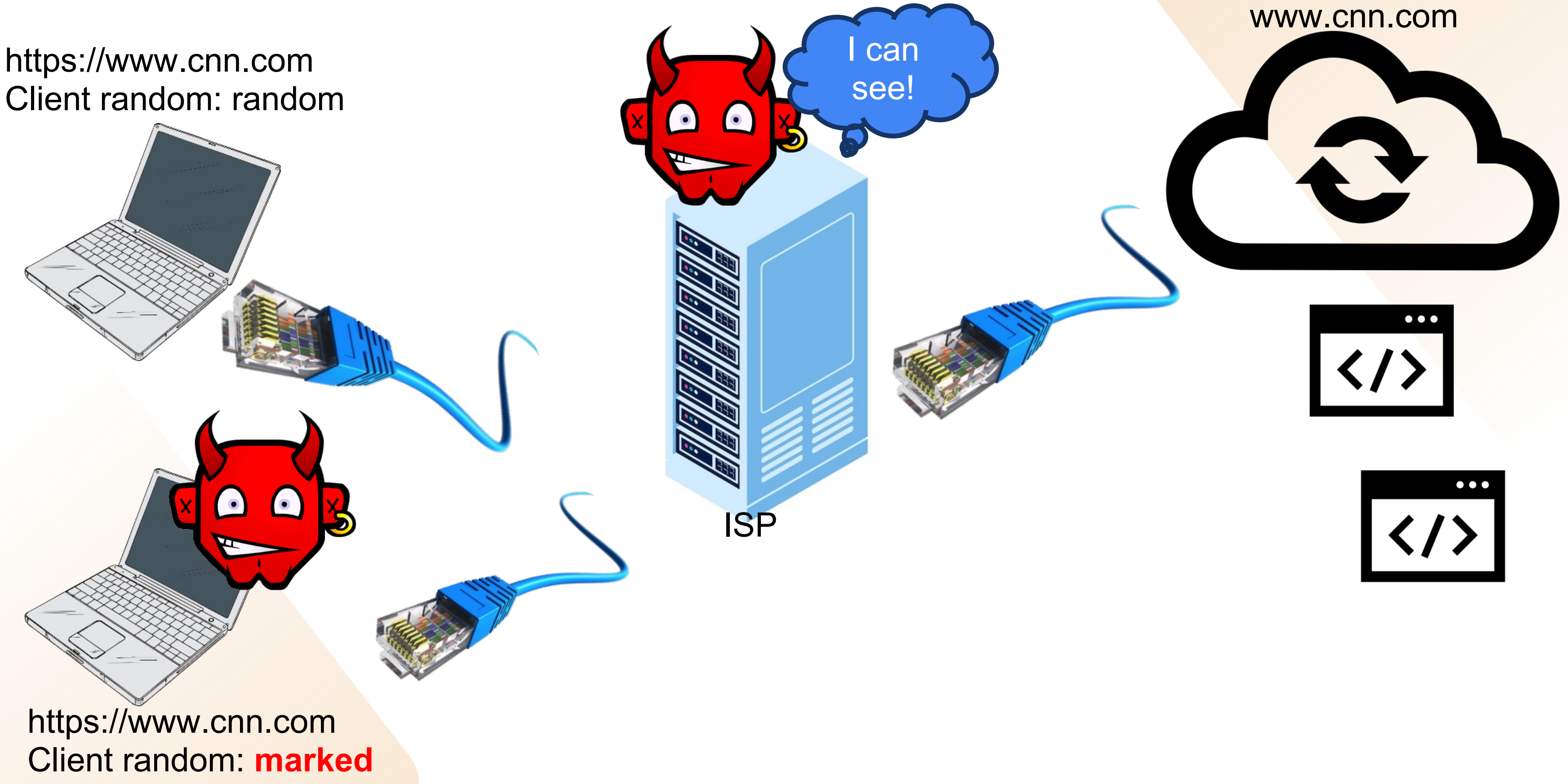
https://blog.cloudflare.com/keyless-ssl-the-nitty-gritty-technical-details

Ephemeral Diffie Hellman (EDH)

- EDH provides Perfect Forward Secrecy to TLS
- Provided the DH private parameter (“secret color”) remains secret…
- But DH parameter is also created with the, now patched, PRNG!
- 😱 Reductor attackers could probably passively eavesdrop! 😱



The Reductor MITM: passive eavesdropper!

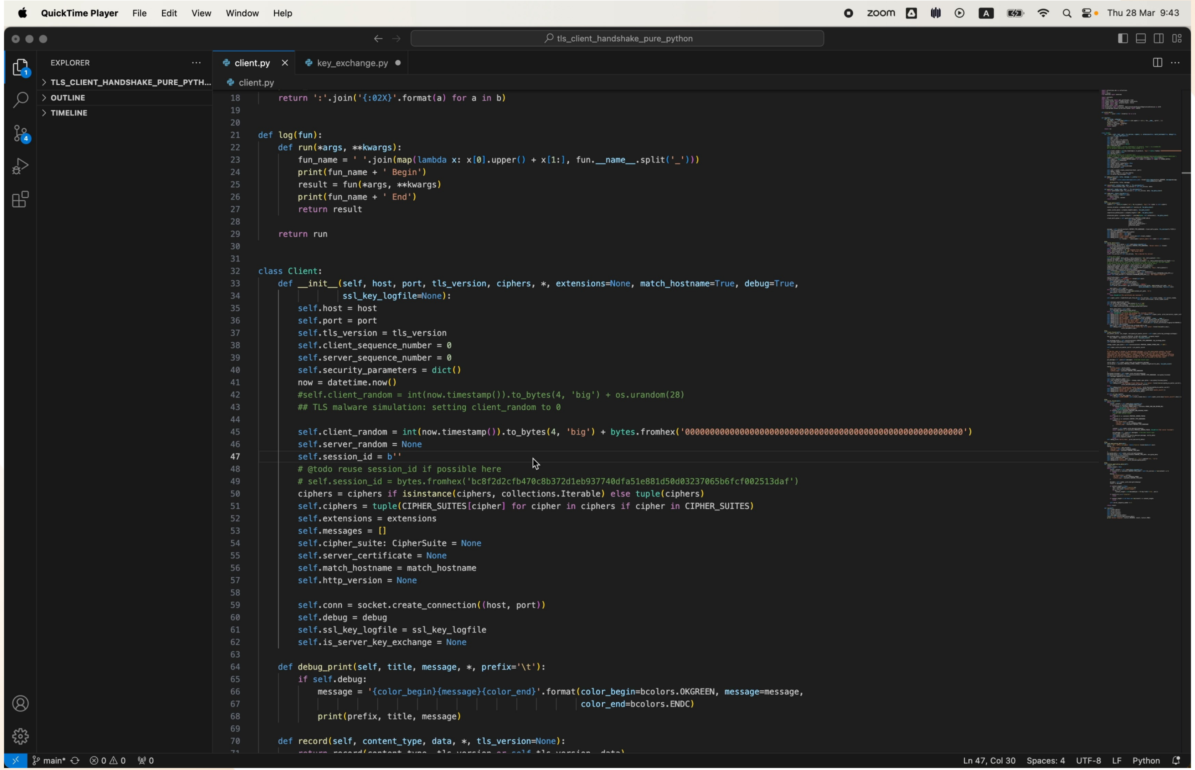


DEMO!

Demo recipe

1. Use our modified TLS client github.com/ZenGo-X/tls_client_handshake_pure_python to patch
   a. Client Random
   b. DH parameter
2. Connect with our modified client via TLS to a well known website
3. Record the encrypted traffic of this connection using Wireshark PCAP
4. Use our tool https://github.com/ZenGo-X/TLS-masterkey-recovery key to compute the masterkey using
   a. inputs
      i. Server parameters in plaintext, as obtained from PCAP
         1. Server random
         2. Server DH public key
      ii. The predetermined Client parameters
         1. Client Random (as obtained from PCAP)
         2. Client DH private key
   b. Save the masterkey output in the standard SSLKEYLOGFILE format
5. Feed this masterkey file to Wireshark to successfully decrypt the traffic
6. WIN!

Demo!

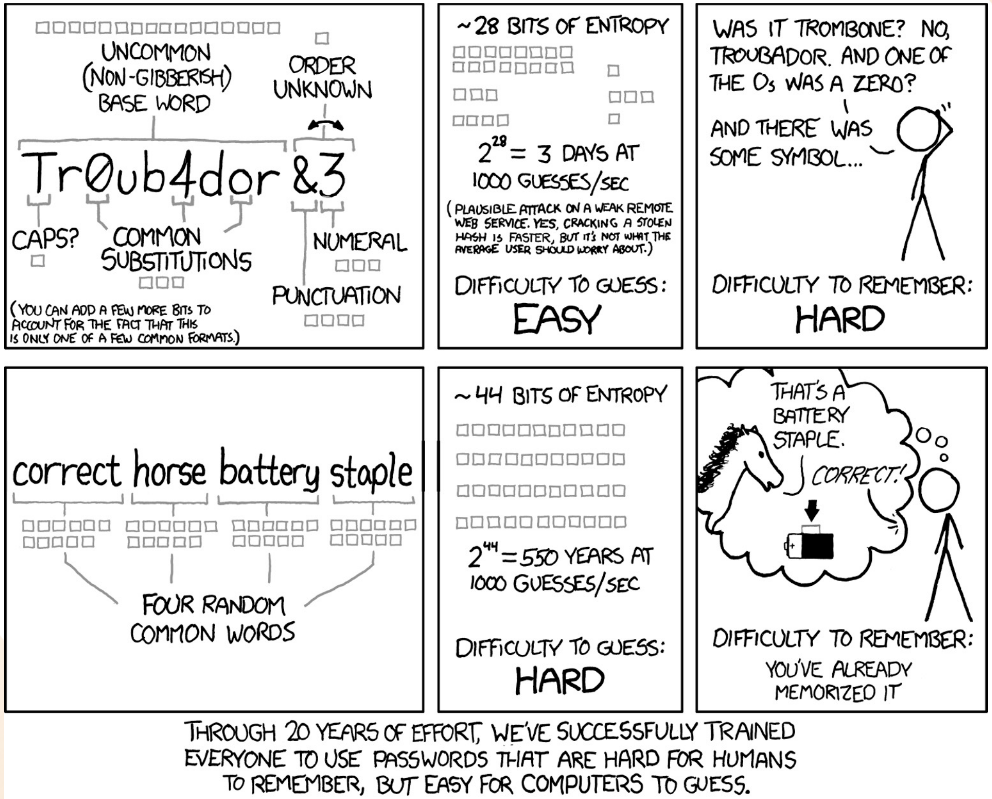

QuickTime Player File Edit View Window Help
Thu 28 Mar 9:43
tls_client_handshake_pure_python
EXPLORER
TLS_CLIENT_HANDSHAKE_PURE_PYTH...
OUTLINE
TIMELINE
client.py
key_exchange.py
return ':'.join('{:02X}'.format(a) for a in b)
def log(fun):
def run(*args, **kwargs):
fun_name = ' '.join(map(lambda x: x[0].upper() + x[1:], fun.__name__.split('_')))
print(fun_name + ' Begin')
result = fun(*args, **kwargs)
print(fun_name + ' End')
return result
return run
class Client:
def __init__(self, host, port, tls_version, ciphers, *, extensions=None, match_hostname=True, debug=True,
ssl_key_logfile=None):
self.host = host
self.port = port
self.tls_version = tls_version
self.client_sequence_number = 0
self.server_sequence_number = 0
self.security_parameters = dict()
now = datetime.now()
#self.client_random = int(now.timestamp()).to_bytes(4, 'big') + os.urandom(28)
TLS malware simulation: setting client_random to 0
self.client_random = int(now.timestamp()).to_bytes(4, 'big') + bytes.fromhex('00000000000000000000000000000000000000000000000000000000')
self.server_random = None
self.session_id = b''
# @todo reuse session_id if possible here
# self.session_id = bytes.fromhex('bc8f2d2cfb470c8b372d1eb937740dfa51e881d50d03237065b6fcf002513daf')
ciphers = ciphers if isinstance(ciphers, collections.Iterable) else tuple(ciphers)
self.ciphers = tuple(CIPHER_SUITES[cipher] for cipher in ciphers if cipher in CIPHER_SUITES)
self.extensions = extensions
self.messages = []
self.cipher_suite: CipherSuite = None
self.server_certificate = None
self.match_hostname = match_hostname
self.http_version = None
self.conn = socket.create_connection((host, port))
self.debug = debug
self.ssl_key_logfile = ssl_key_logfile
self.is_server_key_exchange = None
def debug_print(self, title, message, *, prefix='\t'):
if self.debug:
message = '{color_begin}{message}{color_end}'.format(color_begin=bcolors.OKGREEN, message=message,
color_end=bcolors.ENDC)
print(prefix, title, message)
def record(self, content_type, data, *, tls_version=None):
main*
Ln 47, Col 30 Spaces: 4 UTF-8 LF Python

Some (additional) observations

- Bad randomness is so undetectable that we are not even sure what the attackers have done
- Attackers are even more stealthy now
  - Passiveness is the ultimate stealth mode
- PFS is not always better than no PFS

Solving bad randomness

Bad solution: Human generated randomness



Human generated randomness in the wild

- AKA “brain” wallets
- Entropy is generated from a passphrase
- DEF CON 23 (2012) - Ryan Castellucci - Cracking CryptoCurrency Brainwallets
  - ➔ https://www.youtube.com/watch?v=foil0hzl4Pg
- Found 733 BTC in 2012 ➔ ~$50M in 2024
- “Down the Rabbit-Hole”: held about 85 BTC in July 2012

“

Humans are not a good source of entropy

Bitcoin Wiki https://en.bitcoin.it/wiki/Brainwallet

Removing the need of randomness

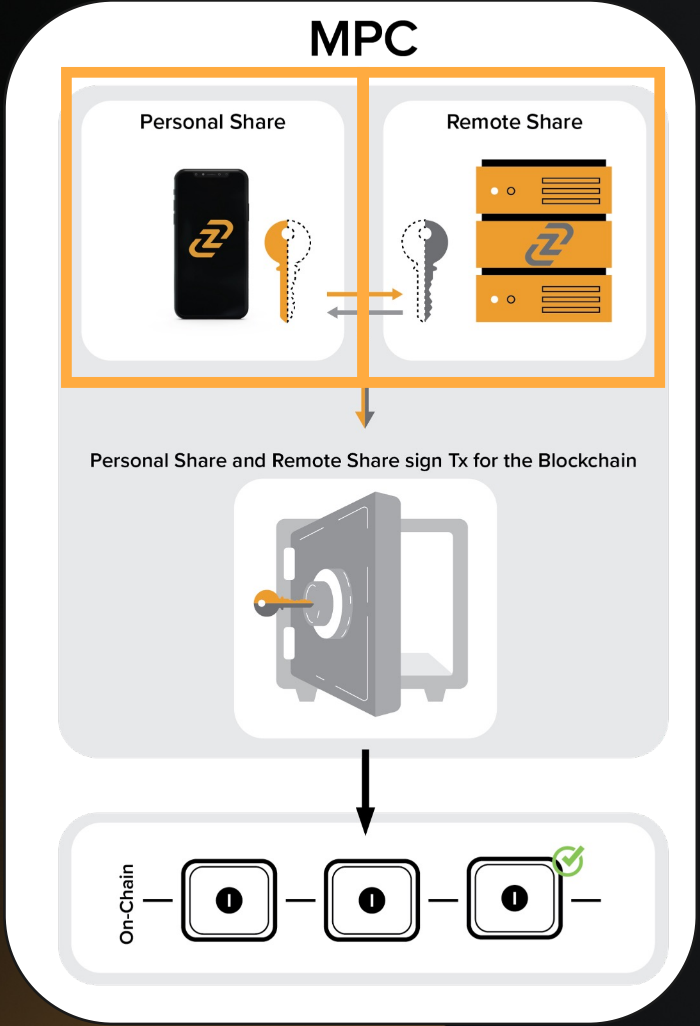
- Reusing existing good randomness
  - Deterministic Nonce (RFC6979)
    - HMAC-SHA256(private_key, message)
  - NAXOS trick (draft-irtf-cfrg-randomness-improvements-10.html)
    - Mix server long term key with entropy
- See also James P. Hughes, Whitfield Diffie: “The Challenges of IoT, TLS, and Random Number Generators in the Real World”
  - https://queue.acm.org/detail.cfm?id=3546933

Protecting the PRNG itself

- Treat PRNG as the most critical part of the system
  - E.g. PRNG protection in hardware
- Helpful, yet limited
  - The PRNG is still single point of failure
- What if we could have it distributed?
  - We can do it with Multi-Party Computation
    - https://drand.love/

Multi-Party computation (MPC) for ECDSA

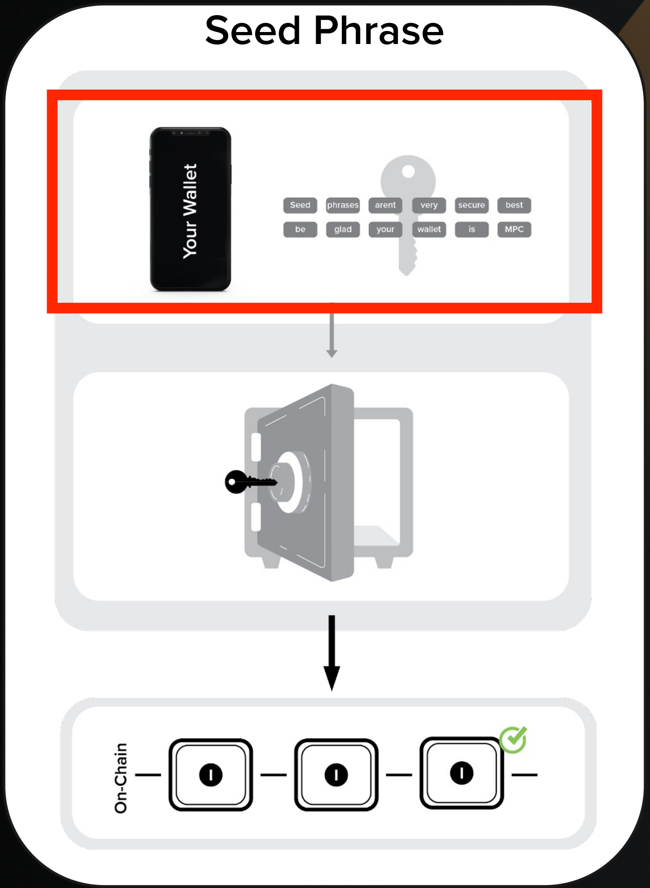
- Key generation is distributed
    - ➔ Bad randomness of a single party still create a random key
- Signing is distributed
    - ➔ Bad randomness of a single party still create a random nonce
- Our implementation
    - ➔ https://github.com/ZenGo-X/gotham-city
    - ➔ Blogs

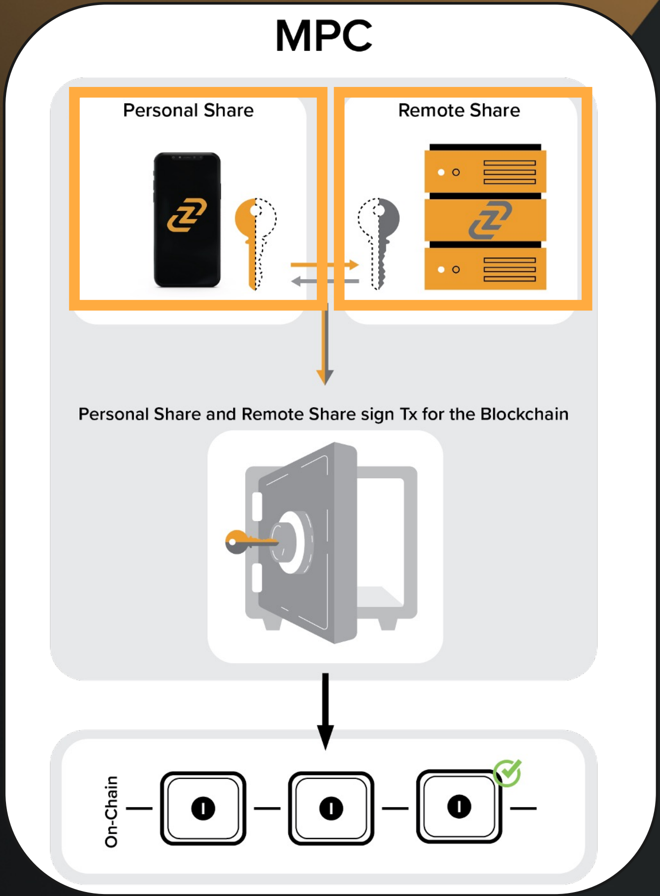


MPC wallets

- No Single Point of Failure!
- Key generation is distributed
  - ➔ Resilient against malware key theft
  - ➔ Resilient against bad randomness
- Signing is distributed
  - ➔ Resilient against malware key theft
  - ➔ Resilient against bad randomness
- Blockchain is unaware
  - ➔ Signature looks the same

Seed Phrase vs. MPC



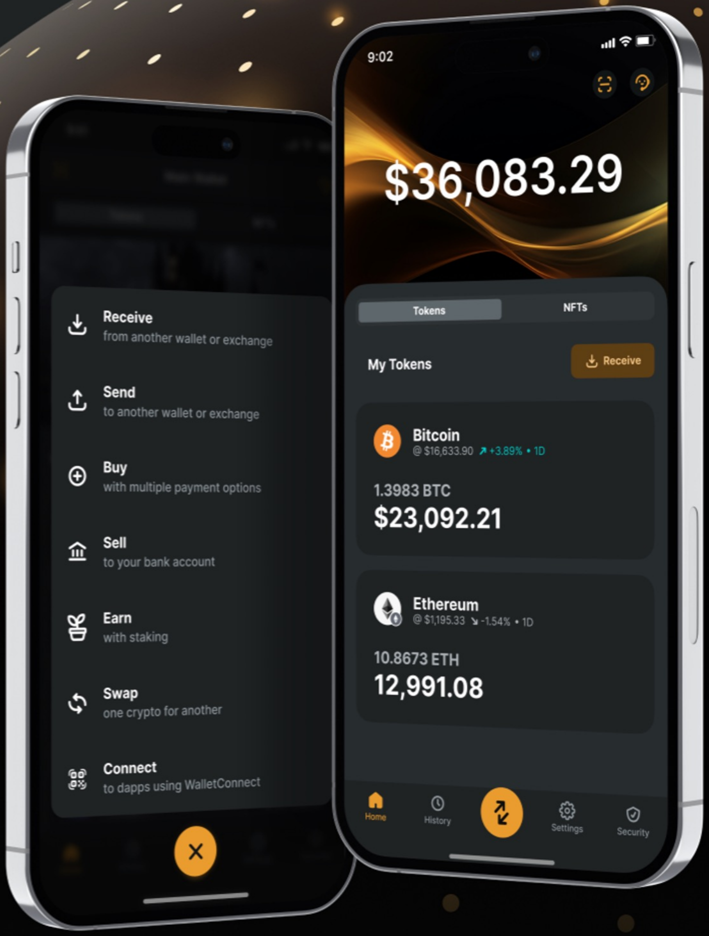


Outro

Takeaways

- Bad randomness is indeed crypto’s perfect crime
- Exploited in the wild
  - ➔ APT for TLS
  - ➔ Bitcoin dark forest attackers
  - ➔ Ethereum dark forest attackers
- Solutions:
  - ➔ Protect PRNG
  - ➔ Remove unnecessary randomness requirements
  - ➔ Use MPC to avoid Single Point of Failure

zengo



Receive
from another wallet or exchange
Send
to another wallet or exchange
Buy
with multiple payment options
Sell
to your bank account
Earn
with staking
Swap
one crypto for another
Connect
to dapps using WalletConnect
9:02
$36,083.29
Tokens
NFTs
My Tokens
Receive
Bitcoin
@ $16,633.90 +3.89% • 1D
1.3983 BTC
$23,092.21
Ethereum
@ $1,195.33 -1.54% • 1D
10.8673 ETH
12,991.08
Home
History
Settings
Security