The Master Key

segher <38c3@segher.eu>
Wanda <wanda@phinode.net>

20241228

HDCP MASTER KEY (MIRROR THIS TEXT!)

This is a forty times forty element matrix of fifty-six bit
hexadecimal numbers.

To generate a source key, take a forty-bit number that (in
binary) consists of twenty ones and twenty zeroes; this is
the source KSV. Add together those twenty rows of the matrix
that correspond to the ones in the KSV (with the lowest bit
in the KSV corresponding to the first row), taking all elements
modulo two to the power of fifty-six; this is the source
private key.

To generate a sink key, do the same, but with the transposed
matrix.

```
6692d179032205 b4116a96425a7f ecc2ef51af1740 959d3b6d07bce4 fa9f2af29814d9
82592e77a204a8 146a6970e3c4a1 f43a81dc36eff7 568b44f60c79f5 bb606d7fe87dd6
1b91b9b73c68f9 f31c6aeef81de6 9a9cc14469a037 a480bc978970a6 997f729d0a1a39
b3b9accda43860 f9d45a5bf64a1d 180a1013ba5023 42b73df2d33112 851f2c4d21b05e
2901308bbd685c 9fde452d3328f5 4cc518f97414a8 8fca1f7e2a0a14 dc8bdbb12e2378
672f11cedf36c5 f45a2a00da1c1d 5a3e82c124129a 084a707eadd972 cb45c81b64808d
07ebd2779e3e71 9663e2beeee6e5 25078568d83de8 28027d5c0c4e65 ec3f0fc32c7e63
1d6b501ae0f003 f5a8fcecb28092 854349337aa99e 9c669367e08bf1 d9c23474e09f70

3c901d46bada9a 40981ffcfa376f a4b686ca8fb039 63f2ce16b91863 1bade89cc52ca2
4552921af8efd2 fe8ac96a02a6f9 9248b8894b23bd 17535dbff93d56 94bdc32a095df2
cd247c6d30286e d2212f9d8ce80a dc55bdc2a6962c bcabf9b5fcbe6f c2cfc78f5fdafa
80e32223b9feab f1fa23f5b0bf0d ab6bf4b5b698ae d960315753d36f 424701e5a944ed
10f61245ebe788 f57a17fc53a314 00e22e88911d9e 76575e18c7956e c1ef4eee022e38
f5459f177591d9 08748f861098ef 287d2c63bd809e e6a28a6f5d000c 7ae5964a663c1b
0f15f7167f56c6 d6c05b2bbe8800 544a49be026410 d9f3f08602517f 74878dc02827f7
d72ef3ea24b7c8 717c7afc0b55a5 0be2a582516d08 202ded173a5428 9b71e35e45943f

9e7cd2c8789c99 1b590a91f1cffd 903dca7c36d298 52ad58ddcc1861 56dd3acba0d9c5
c76254c1be9ed1 06ecb6ae8ff373 cfcc1afcbc80a4 30eba7ac19308c d6e20ae760c986
c0d1e59db1075f 8933d5d8284b92 9280d9a3faa716 8386984f92bfd6 be56cd7c4bfa59
16593d2aa598a6 d62534326a40ee 0c1f1919936667 acbaf0eefdd395 36dbfdbf9e1439
0bd7c7e683d280 54759e16cfd9ea cac9029104bd51 436d1dca1371d3 ca2f808654cdb2
7d6923e47f97b5 70e256b741910c 7dd466ed5fff2e 26bec4a28e8cc4 5754ea7219d4eb
75270aa4d3cc8d e0ae1d1897b7f4 4fe5663e8cb342 05a80e4a1a950d 66b4eb6ed4c99e
3d7e9d469c6165 81677af04a2e15 ada4be60bc348d dfdfbbad739248 98ad5986f3ca1f
```

What is HDCP?

Why do we care?

What can we do about it?

I found a firmware update with a key pair in it

I found a firmware update with a key pair in it
And then another

I found a firmware update with a key pair in it
And then another
And then another

I found a firmware update with a key pair in it
And then another
And then another
Seven or eight or nine total

I found a firmware update with a key pair in it
And then another
And then another
Seven or eight or nine total

But that is not enough, we need forty or a bit more

So, where to get more keypairs?

So, where to get more keypairs?
We could get many Mac laptops

So, where to get more keypairs?
We could get many Mac laptops
But in the end we could not easily get the keys out

So, new approach

So, new approach
Keys are stored close to the video driver

So, new approach
Keys are stored close to the video driver
So let's try to get stuff from there!

nvidia comes in

- In 2010, I (Wanda) was working on reverse engineering nvidia hardware as part of project nouveau
- At one point I ended up reversing mysterious blobs of on-GPU firmware in a custom ISA, related to video decoding
- We later learned this is their in-house Falcon architecture
- As in turns out, one of the Falcon cores has a secure co-processor, and one of its duties is decrypting the HDCP keys and sending them to the display unit

The Falcon architecture

- Falcon (FAst Logic CONtroller) is a generic RISC-ish CPU architecture made in-house by nvidia, used as their default go-to management core
  - ISA is unremarkable
  - Uses dedicated tiny SRAM for code and data segments (~5kiB)
  - Has DMA capabilities to and from VRAM
- We looked at the very first version of the architecture (v0), used in only three devices: G98 (low-end GPU), MCP77 (motherboard chipset with integrated GPU), MCP79 (another chipset with integrated GPU)

The Falcon architecture

- The GPU contains four Falcon cores, managing various video decoding engines:
  - MSBSP: video bitstream processing
  - MSPDEC: picture decoding
  - MSPPP: picture post-processing
  - SEC: DRM stuff
- The SEC Falcon has a Secure CoProcessor (SCP), which extends the Falcon ISA with AES opcodes, code signing, and key storage

Falcon and HDCP

- HDCP is mostly handled by the display unit, which is hardwired logic
- The HDCP key is large (285 bytes) and is stored encrypted in an I²C EEPROM
- On IGPs, the encrypted key is stored in main BIOS flash instead
- On startup, a signed bit of code is run on SEC that decrypts the HDCP key and sends it over a dedicated bus to the display unit

The SCP coprocessor — main features

- The Falcon can run in secure or non-secure mode
- 8× 128-bit registers for key and plaintext/ciphertext block storage and working area
  - Each register has a 5-bit ACL (readable/writable/usable-as-key by non-secure code, readable/usable-as-key by secure code)
- AES encryption/decryption, CMAC computation, AND, XOR, MOV instructions

The SCP coprocessor — main features

- 63× 128-bit mask ROM secret storage, identical across all GPUs
  - Secrets likewise have ACLs
- 1× 40-bit fuse key storage, per-GPU
- Streaming DMA to and from:
  - Falcon code segment
  - Falcon data segment
  - VRAM and system RAM
- A true RNG (not used for our purposes)

The SCP coprocessor — secure mode entry

- Upload the secure-mode code into code RAM, mark it as secure using special registers
- Put the MAC into one of the SCP registers
- Jump to the code
- Instead of running the code, Falcon will jump to the secret ROM that validates the signature and, if correct, jumps to the code in secure mode
- If the signature is incorrect, you get a trap

The HDCP firmware

- The code is not encrypted so we can just reverse it
- However, the HDCP keys are (unsurprisingly) decrypted using the mask keys that cannot be read from non-secure mode
- For unclear reasons, the decryption key is also encrypted using the decryptor code’s MAC
  - as it later turned out, the resulting decryption key is actually identical to the encrypting mask secret???
- The EEPROM decryption key was not tied to the particular device

The SCP coprocessor — security flaws

- We cannot read the secrets to “normal” storage, but we can MOV them to registers just fine, they just retain their “unreadable” ACL
- We can even perform AND/XOR operations — the result just inherits the most restrictive ACL
- … and we can try entering secure mode without being actually able to read the MAC we are presenting

The SCP coprocessor — secret extraction

The extraction algorithm is really simple:

- Load 1 << i to a crypto register
- AND it with a secret
- XOR it with a known-good MAC of some random piece of nvidia code
- Try entering secure mode
- If it still runs, bit i of the secret was a 0

This even lets us read secrets marked as unreadable from secure mode.

SCP security for secret storage completely broken at this point…

The SCP coprocessor — ACE in secure mode

- … but we ran into a problem with an unknown (at the time) opcode in the HDCP blob in the code used to derive the decryption key
- The opcode didn’t seem to do anything at all in non-secure mode
  - But if we actually tried replicating the blob’s logic with a NOP there, it didn’t decrypt right
- … so we needed to figure out a way to run our code in secure mode to reverse the missing opcode

The SCP coprocessor — ACE in secure mode

- We had dumped all secrets at this point, and tried various AES-based MAC algorithms, but found no match
- As it turns out, if you fail secure mode entry, the correct MAC is just left in one of the SCP registers
- 
- We can read it using the previous trick
- Or just MOV it to the right register and retry secure mode entry

The SCP coprocessor — the missing bits

- As it turns out, the unknown opcode was just “encrypt this block using the current code’s MAC as a key” — it only added obscurity, not real security
- Also, in this particular code, the result of this operation was just equal to the originally used mask key

… and with that, we had the capability to dump the full HDCP key from any G98-based GPU or MCP77/MCP79-based motherboard

The SCP coprocessor — followup

- Years later, we found out the MAC algorithm is actually an obscure variant of Davies-Meyer construction (turning AES into a hash function) combined with CMAC
- The flaws we used have been fixed on the next-generation (v3) Falcon
  - Not our doing, v3 was released before we ever looked at Falcon
  - This is why we had to target a very narrow range of GPUs

The SCP coprocessor — followup

- Falcon became relevant again due to its use on Nintendo Switch
  - A great way to attract many hackers!
- Turns out *someone* broke the root of trust in the ARM bootROM, so Nintendo tried to recover by using Tegra's SEC2 as the new root of trust
- This time it fell to blind ROP in the Falcon secret ROM (use DMA to smash the stack)

Linear algebra

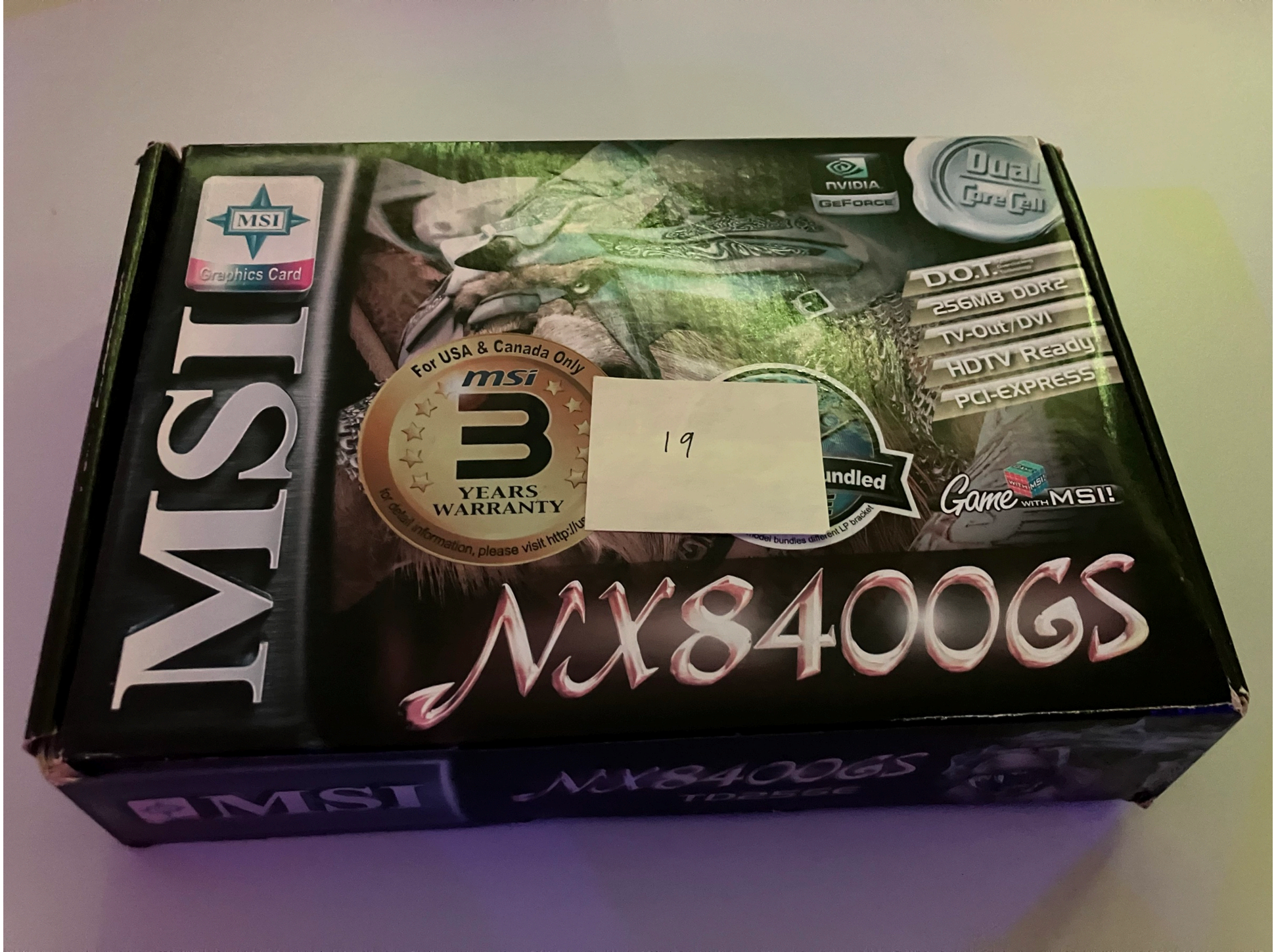
segher 20241228

Bushing decided to buy 45 or so cards, from Newegg

When returning the cards, they pay you back day price

When returning the cards, they pay you back day price
We made money!

When returning the cards, they pay you back day price
We made money!
(This is not a viable business plan though)


MSI
Graphics Card
NVIDIA
GeForce
Dual
Core Cell
D.O.T.
256MB DDR2
TV-Out/DVI
HDTV Ready
PCI-EXPRESS
For USA & Canada Only
msi
3
YEARS
WARRANTY
for detail information, please visit http://us
19
Game WITH MSI!
MSI
NX8400GS
MSI
NX8400GS

msi
N1996

So now we had enough keys to do whatever we wanted

The sink (like, a display) uses its own private key together with the source's public key (its KSV) to derive a shared secret

The sink (like, a display) uses its own private key together with the source's public key (its KSV) to derive a shared secret

And the source does it the other way around, and the HDCP standard tells us that will give us the same shared secret

The sink (like, a display) uses its own private key together with the source's public key (its KSV) to derive a shared secret

And the source does it the other way around, and the HDCP standard tells us that will give us the same shared secret

"Key selection vector" means to take the 20 (out of 40) columns that have a 1 in the KSV

This is the same as taking their internal product; written as just a multiplication

This is the same as taking their internal product; written as just a multiplication

Taking linear combinations of public keys you can make other public keys, and the same linear combination of the corresponding private keys is the private key you need for that

Taking linear combinations of public keys you can make other public keys, and the same linear combination of the corresponding private keys is the private key you need for that

This is all you do for Gaussian elimination, so that works just fine, we can solve the whole system of equations, invert the whole matrix, derive the "master key"!

This is all you do for Gaussian elimination, so that works just fine, we can solved the whole system of equations, invert the whole matrix, derive the "master key"!

Well, almost

This is all you do for Gaussian elimination, so that works just fine, we can solved the whole system of equations, invert the whole matrix, derive the "master key"!
Well, almost

We are working in $\mathbb{Z}/2^{56}\mathbb{Z}$, this isn't a field, the vectors are not in a vector space, they are in a more general module

We are working in $\mathbb{Z}/2^{56}\mathbb{Z}$, this isn't a field, the vectors are not in a vector space, they are in a more general module

Everything just works as expected though, except there isn't a unique solution

We are working in $\mathbb{Z}/2^{56}\mathbb{Z}$, this isn't a field, the vectors are not in a vector space, they are in a more general module
Everthing just works as expected though, except there isn't a unique solution

And, there isn't enough data to fully solve everything anyway: all valid KSVs have 20 (out of 40) bits equal to 1, which means you cannot make any linear combination of them that have just one bit set

And, there isn't enough data to fully solve everything anyway: all valid KSVs have 20 (out of 20) bits equal to 1, which means you cannot make any linear combination of them that have just one bit set

But that doesn't matter so much

And, there isn't enough data to fully solve everything anyway: all valid KSVs have 20 (out of 20) bits equal to 1, which means you cannot make any linear combination of them that have just one bit set
But that doesn't matter so much

We cannot find a unique solution, but we can find one that works!

We cannot find a unique solution, but we can construct something that works!

In principle it is possible that from the data we reconstruct it can be determined what key pairs we used to reconstruct it, and we cannot have that, that is basic opsec
So I filled in random noise for all data that cannot be uniquely derived

```
srand(42)
```

And, any KSV has exactly 20 bits set, and 20 is a multiple of 4

So we can add whatever number we want to the top 2 bits of all entries in any column and it will still work

Same for row, everything is symmetric

We can also add the same number in the top 4 bits of all entries in the whole matrix: 20x20=400 is a multiple of 16

Leaking it

We contemplated various avenues to get the thing into the public's hands without showing it was us

We contemplated various avenues to get the thing into the public's hands without showing it was us

We contacted wikileaks, but this isn't their mission really, they didn't want to do it. Fair enough; well we could try :-)


twitter
Home Profile Find Peo
IntelGlobalPR
Follow
Lists
@spacerog @freaklabs @wired @adafruit HDCP key released: http://bit.ly/aM84GD (please mirror/RT!) #drm #hdcp #defectivebydesign
8:50 PM Sep 13th via web
@chronicdevteam @G33KatWork @firefoxsux HDCP key released: http://bit.ly/aM84GD (please mirror/RT!) #drm #hdcp #defectivebydesign
8:43 PM Sep 13th via web
@daeken @trent_reznor @leolaporte HDCP key released: http://bit.ly/aM84GD (please mirror/RT!) #drm #hdcp #defectivebydesign
8:35 PM Sep 13th via web
@0xFFFFFFFE @mdowd @eff @enkido @dmolnar HDCP key released: http://bit.ly/aM84GD (please mirror/RT!) #drm #hdcp

HDCP MASTER KEY (MIRROR THIS TEXT!)

This is a forty times forty element matrix of fifty-six bit
hexadecimal numbers.

To generate a source key, take a forty-bit number that (in
binary) consists of twenty ones and twenty zeroes; this is
the source KSV. Add together those twenty rows of the matrix
that correspond to the ones in the KSV (with the lowest bit
in the KSV corresponding to the first row), taking all elements
modulo two to the power of fifty-six; this is the source
private key.

To generate a sink key, do the same, but with the transposed
matrix.

6692d179032205 b4116a96425a7f ecc2ef51af1740 959d3b6d07bce4 fa9f2af29814d9
82592e77a204a8 146a6970e3c4a1 f43a81dc36eff7 568b44f60c79f5 bb606d7fe87dd6
1b91b9b73c68f9 f31c6aeef81de6 9a9cc14469a037 a480bc978970a6 997f729d0a1a39
b3b9accda43860 f9d45a5bf64a1d 180a1013ba5023 42b73df2d33112 851f2c4d21b05e
2901308bbd685c 9fde452d3328f5 4cc518f97414a8 8fca1f7e2a0a14 dc8bdbb12e2378
672f11cedf36c5 f45a2a00da1c1d 5a3e82c124129a 084a707eadd972 cb45c81b64808d
07ebd2779e3e71 9663e2beeee6e5 25078568d83de8 28027d5c0c4e65 ec3f0fc32c7e63
1d6b501ae0f003 f5a8fcecb28092 854349337aa99e 9c669367e08bf1 d9c23474e09f70

3c901d46bada9a 40981ffcfa376f a4b686ca8fb039 63f2ce16b91863 1bade89cc52ca2
4552921af8efd2 fe8ac96a02a6f9 9248b8894b23bd 17535dbff93d56 94bdc32a095df2
cd247c6d30286e d2212f9d8ce80a dc55bdc2a6962c bcabf9b5fcbe6f c2cfc78f5fdafa
80e32223b9feab f1fa23f5b0bf0d ab6bf4b5b698ae d960315753d36f 424701e5a944ed
10f61245ebe788 f57a17fc53a314 00e22e88911d9e 76575e18c7956e c1ef4eee022e38
f5459f177591d9 08748f861098ef 287d2c63bd809e e6a28a6f5d000c 7ae5964a663c1b
0f15f7167f56c6 d6c05b2bbe8800 544a49be026410 d9f3f08602517f 74878dc02827f7
d72ef3ea24b7c8 717c7afc0b55a5 0be2a582516d08 202ded173a5428 9b71e35e45943f

9e7cd2c8789c99 1b590a91f1cffd 903dca7c36d298 52ad58ddcc1861 56dd3acba0d9c5
c76254c1be9ed1 06ecb6ae8ff373 cfcc1afcbc80a4 30eba7ac19308c d6e20ae760c986
c0d1e59db1075f 8933d5d8284b92 9280d9a3faa716 8386984f92bfd6 be56cd7c4bfa59
16593d2aa598a6 d62534326a40ee 0c1f1919936667 acbaf0eefdd395 36dbfdbf9e1439
0bd7c7e683d280 54759e16cfd9ea cac9029104bd51 436d1dca1371d3 ca2f808654cdb2
7d6923e47f97b5 70e256b741910c 7dd466ed5fff2e 26bec4a28e8cc4 5754ea7219d4eb
75270aa4d3cc8d e0ae1d1897b7f4 4fe5663e8cb342 05a80e4a1a950d 66b4eb6ed4c99e
3d7e9d469c6165 81677af04a2e15 ada4be60bc348d dfdfbbad739248 98ad5986f3ca1f

“Intel confirmed last week that the master key published online was genuine and is now investigating if the key was leaked or cracked”