Practical Cryptographic Systems

Hardware Security & Tamper Resistance

Matthew Green

News?

News?

Why does Claude Speak Byzantine Music Notation?

31st of March 2025

A Caesar cipher is a reasonable transformation for a transformer to learn in its weights, given that a specific cipher offset occurs often enough in its training data. There will be some hidden representation of the input tokens' spelling, and this representation could be used to shift letters onto other letters in even a single attention head. Most frontier models can fluently read and write a Caesar cipher on ASCII text, with offsets that presumably occur in their training data, like 1, -1, 2, 3, etc.

As we will shortly see, they can also infer the correct offset on the fly given a short sentence, which is already quite impressive for a single forward pass.

It is also natural that this effect does not generalize to uncommon offsets, because numerical algorithms implemented in the weights are restricted to values in the training distribution.

News?

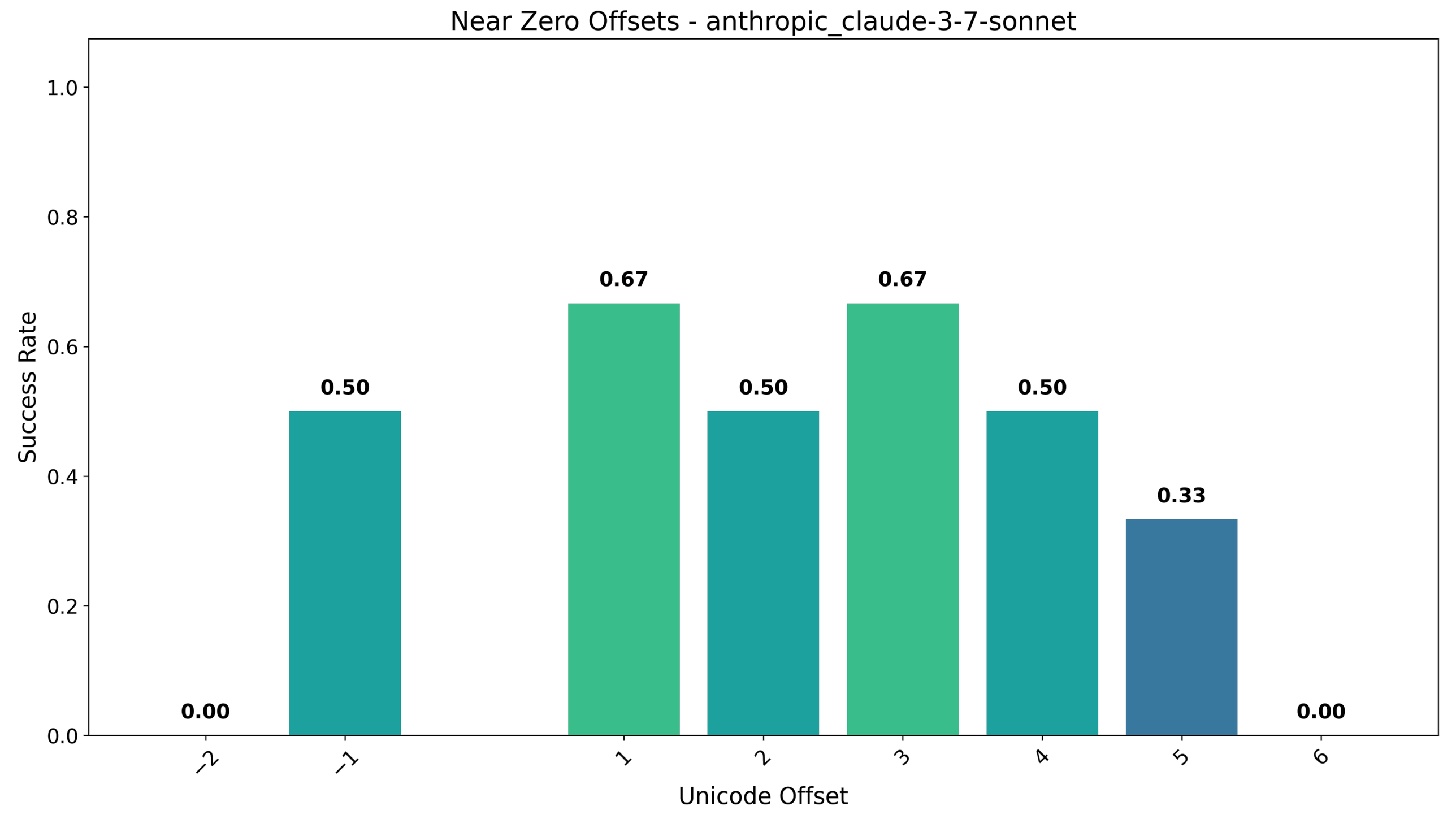
Why does Claude Speak Byzantine Music Notation?

31st of March 2025

A Caesar cipher is a reasonable transformation for a transformer to learn in its weights, given that a specific cipher offset occurs often enough in its training data. There will be some hidden representation of the input tokens' spelling, and this representation could be used to shift letters onto other letters in even a single attention head. Most frontier models can fluently read and write a Caesar cipher on ASCII text, with offsets that presumably occur in their training data, like 1, -1, 2, 3, etc.

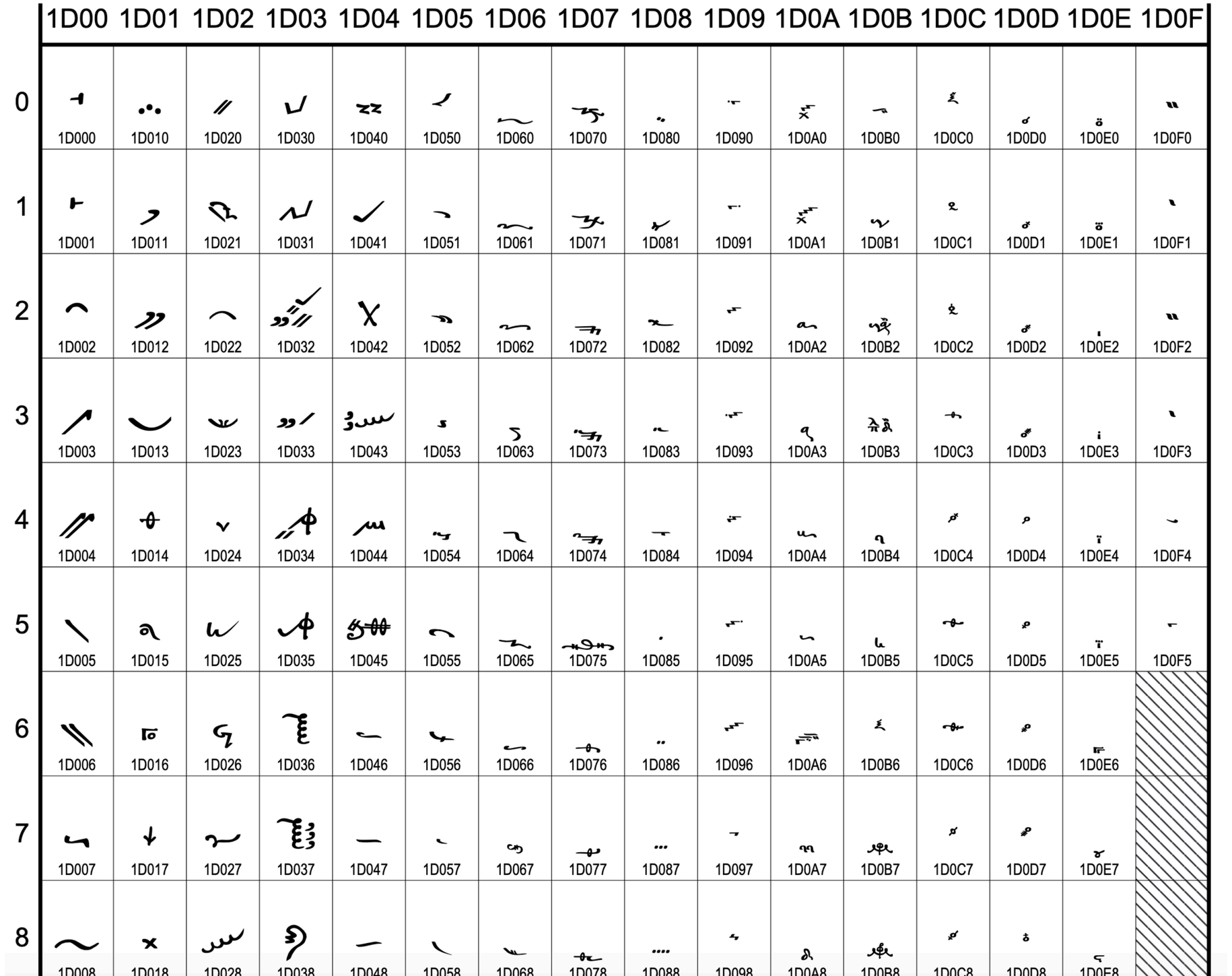
As we will shortly see, they can also infer the correct offset on the fly given a short sentence, which is already quite impressive for a single forward pass.

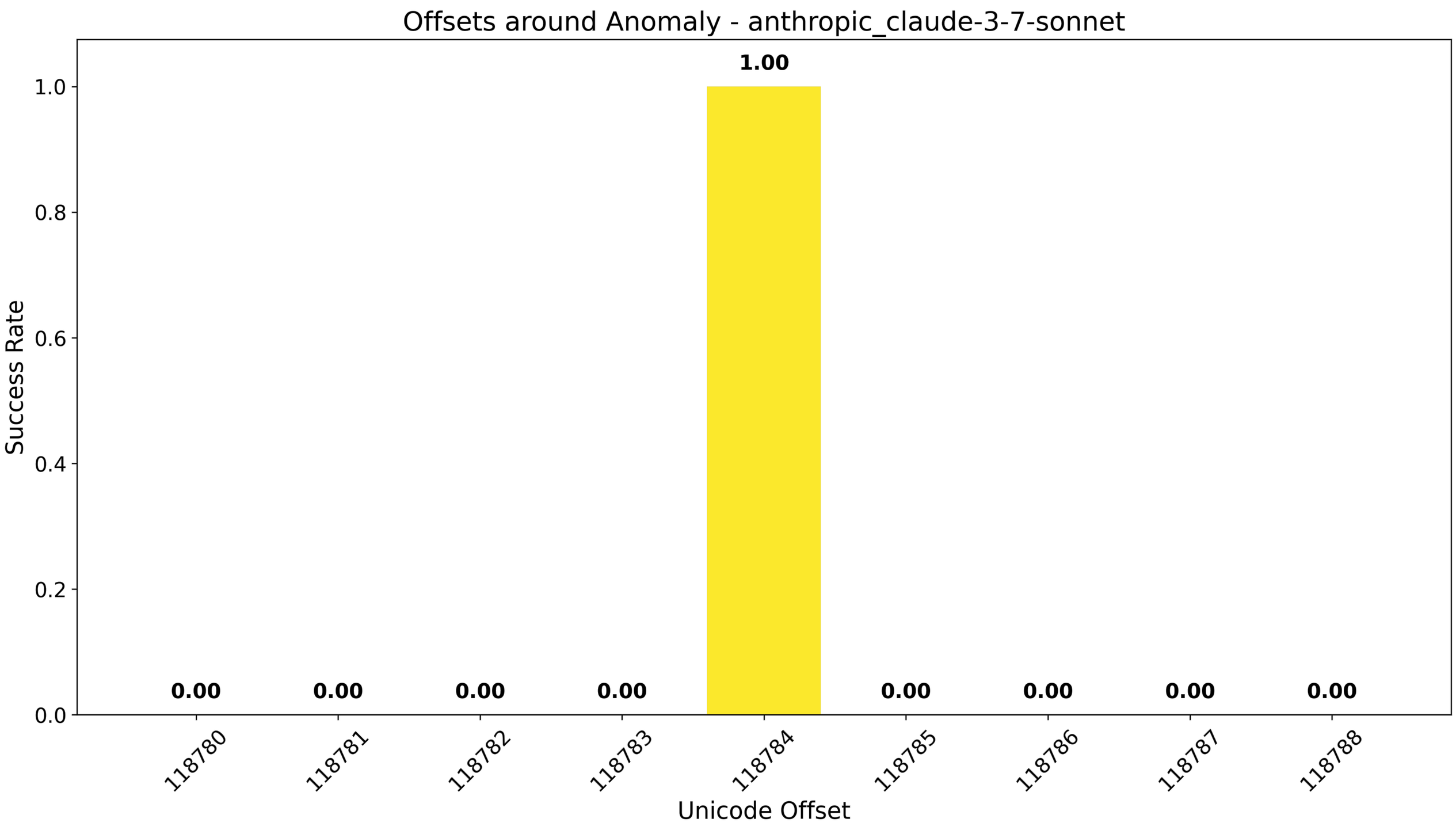
It is also natural that this effect does not generalize to uncommon offsets, because numerical algorithms implemented in the weights are restricted to values in the training distribution.


Near Zero Offsets - anthropic_claude-3-7-sonnet
Success Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.00
0.50
0.67
0.50
0.67
0.50
0.33
0.00
-2
-1
1
2
3
4
5
6
Unicode Offset

This was my understanding at least, until reading Erziev (2025), a description of a phenomenon where many models including Claude and gpt-4o fluently read and write hidden messages in high Unicode ranges, such as the href="https://en.wikipedia.org/wiki/Byzantine_Musical_Symbols">Byzantine music notation Unicode block.

For the specific case of the Byzantine music Unicode block, we can understand the transformation as a Caesar-like cipher in Unicode space, with offset 118784. Using it in the Caesar-like Unicode cipher leads to near-perfect decoding accuracy.


1D00 1D01 1D02 1D03 1D04 1D05 1D06 1D07 1D08 1D09 1D0A 1D0B 1D0C 1D0D 1D0E 1D0F
0 1D000 1D010 1D020 1D030 1D040 1D050 1D060 1D070 1D080 1D090 1D0A0 1D0B0 1D0C0 1D0D0 1D0E0 1D0F0
1 1D001 1D011 1D021 1D031 1D041 1D051 1D061 1D071 1D081 1D091 1D0A1 1D0B1 1D0C1 1D0D1 1D0E1 1D0F1
2 1D002 1D012 1D022 1D032 1D042 1D052 1D062 1D072 1D082 1D092 1D0A2 1D0B2 1D0C2 1D0D2 1D0E2 1D0F2
3 1D003 1D013 1D023 1D033 1D043 1D053 1D063 1D073 1D083 1D093 1D0A3 1D0B3 1D0C3 1D0D3 1D0E3 1D0F3
4 1D004 1D014 1D024 1D034 1D044 1D054 1D064 1D074 1D084 1D094 1D0A4 1D0B4 1D0C4 1D0D4 1D0E4 1D0F4
5 1D005 1D015 1D025 1D035 1D045 1D055 1D065 1D075 1D085 1D095 1D0A5 1D0B5 1D0C5 1D0D5 1D0E5 1D0F5
6 1D006 1D016 1D026 1D036 1D046 1D056 1D066 1D076 1D086 1D096 1D0A6 1D0B6 1D0C6 1D0D6 1D0E6
7 1D007 1D017 1D027 1D037 1D047 1D057 1D067 1D077 1D087 1D097 1D0A7 1D0B7 1D0C7 1D0D7 1D0E7
8 1D008 1D018 1D028 1D038 1D048 1D058 1D068 1D078 1D088 1D098 1D0A8 1D0B8 1D0C8 1D0D8 1D0E8


Offsets around Anomaly - anthropic_claude-3-7-sonnet
Success Rate
1.0
0.8
0.6
0.4
0.2
0.0
1.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
118780
118781
118782
118783
118784
118785
118786
118787
118788
Unicode Offset

The reason that this has any chance of working is the following. At least in most public tokenizers like o200k, addition in certain Unicode ranges commutes with addition in token space. For instance, if we define $\tau : \Sigma^* \to \hat{\Sigma}^*$ as the o200k tokenizer, then in a subset of the Byzantine music notation range (`U+118784` – `U+119029`), a linearity property holds.

$$\tau([\texttt{U+118881} + k]) = [43120, 223, 94 + k], k \in \{0, \ldots, 29\} \setminus \{12\},$$

so all but one of these symbols is mapped to three tokens each, where the first two are the same and can be easily ignored by an attention head, and the third token increments exactly with the Unicode.

The special offset 118784 lands the character "a" on `U+118881`, which is the first in the series of characters that get tokenized in the above arithmetic series. In particular, it is interesting that an offset that is one or two larger does not work. This means that the model has learned a map specifically from the binary range b'\xa1'–b'\xba' to the lowercase ASCII range 97–122. I would believe it if someone told me that this somehow occurs often in the training data, but I cannot think of how exactly.

However, gpt-4o retains some deciphering capability even with an offset one larger, which is weak confirmation that commutativity of addition in the Unicode-token spaces is part of the story. Since Claude handles the deciphering well, we could conclude that its secret tokenizer also handles binary strings with arithmetic increments, like o200k.

So, let me see what I think I understand here:

1. AI models are good at Cæsar-cypher transposition, because it occurs often enough in training models for certain values of the cypher offset. Outside those values, AI doesn't handle the transformations well.

2. Somehow AI models perform this cypher also within high ranges of Unicode, because the characters are encoded as three tokens each, of which the last one encodes the same difference as between alphabetic letters, and so the first two tokens get discarded as irrelevant, meaning that by sheer chance the alphabet maps perfectly via Cæsar-cypher (with a two-token offset) to a specific range of Unicode characters reserved for Byzantine music notation.

3. This is easy to understand for one AI model, because its explicable by chance that the offset between the alphabet and Byzantine music notation should coincide perfectly with two less-significant tokens. It's harder to understand why this works in more than one AI model, though.

yorwba 11 hours ago | parent | next [–]

This is exactly what's happening here. But note that UTF-8 is self-synchronizing, so no encoding of one character contains the encoding of another as a subsequence. Instead, both tag characters and the Byzantine music notation in the article look like "<some prefix bytes> <byte pattern of the corresponding ASCII character + 96>"

They share this property with the Fullwidth Latin block, which does occur in the wild interspersed with Japanese or Chinese text.

yorwba 22 hours ago | parent | next [–]

It's not that surprising that models encode Byzantine Music Notation characters using separate tokens for each UTF-8 byte, since they're unlikely to occur often enough for byte-pair encoding to allocate dedicated multi-byte tokens to them.

What isn't clear to me is where ASCII text with 64 added to every byte occurs in the wild.

immibis 21 hours ago | root | parent | prev | next [–]

Have you checked the UTF-8 encodings of all the typical "Unicode fonts" e.g. circled letters, superscripts and so on?

reply

yorwba 12 hours ago | root | parent | next [–]

Good idea. I checked all alphanumeric characters whose UTF-8 encoding ends with \xa1.

```
  >>> '\N{fullwidth latin capital letter
a}'.encode('utf-8')
  b'\xef\xbc\xa1'
```

A seems like the culprit.

reply

immibis 8 hours ago | root | parent | next [–]

Mystery solved!

reply

Type or Paste

What is Full-Width Character

A full-width character means the character has the same width as a Chinese character, regardless of font choice.

Full-width characters are used in Chinese and Japanese texts. • Unicode: Chinese 中文 • Unicode: Japanese Writing System の

Here is the complete list of full-width characters that is English letters or English punctuation:

０ １ ２ ３ ４ ５ ６ ７ ８ ９

Ａ Ｂ Ｃ Ｄ Ｅ Ｆ Ｇ Ｈ Ｉ Ｊ Ｋ Ｌ Ｍ Ｎ Ｏ Ｐ Ｑ Ｒ Ｓ Ｔ Ｕ Ｖ Ｗ Ｘ Ｙ Ｚ

ａ ｂ ｃ ｄ ｅ ｆ ｇ ｈ ｉ ｊ ｋ ｌ ｍ ｎ ｏ ｐ ｑ ｒ ｓ ｔ ｕ ｖ ｗ ｘ ｙ ｚ

消息，知道革命黨雖然進了城，倒還沒有什麼大異樣。知縣大老爺還是原官，不過改稱了什麼，而且舉人老爺也
件可怕的事是另有幾個不好的革命黨夾在裡面搗亂，第二天便動手剪辮子，聽說那鄰村的航船七斤便¨了道兒，
刻變了計，碰不¨這危險。阿Ｑ本也想進城去尋他的老朋友，一得這消息，也祇得作罷了。

在頂上的逐漸增加起來了，早經說過，最先自然是茂才公，其次便是趙司晨和趙白眼，後來是阿Ｑ。倘在夏天，大家將辮子盤在頭頂
英斷，而在未莊也不能說無關於改革了。

大嚷說，「革命黨來了！」

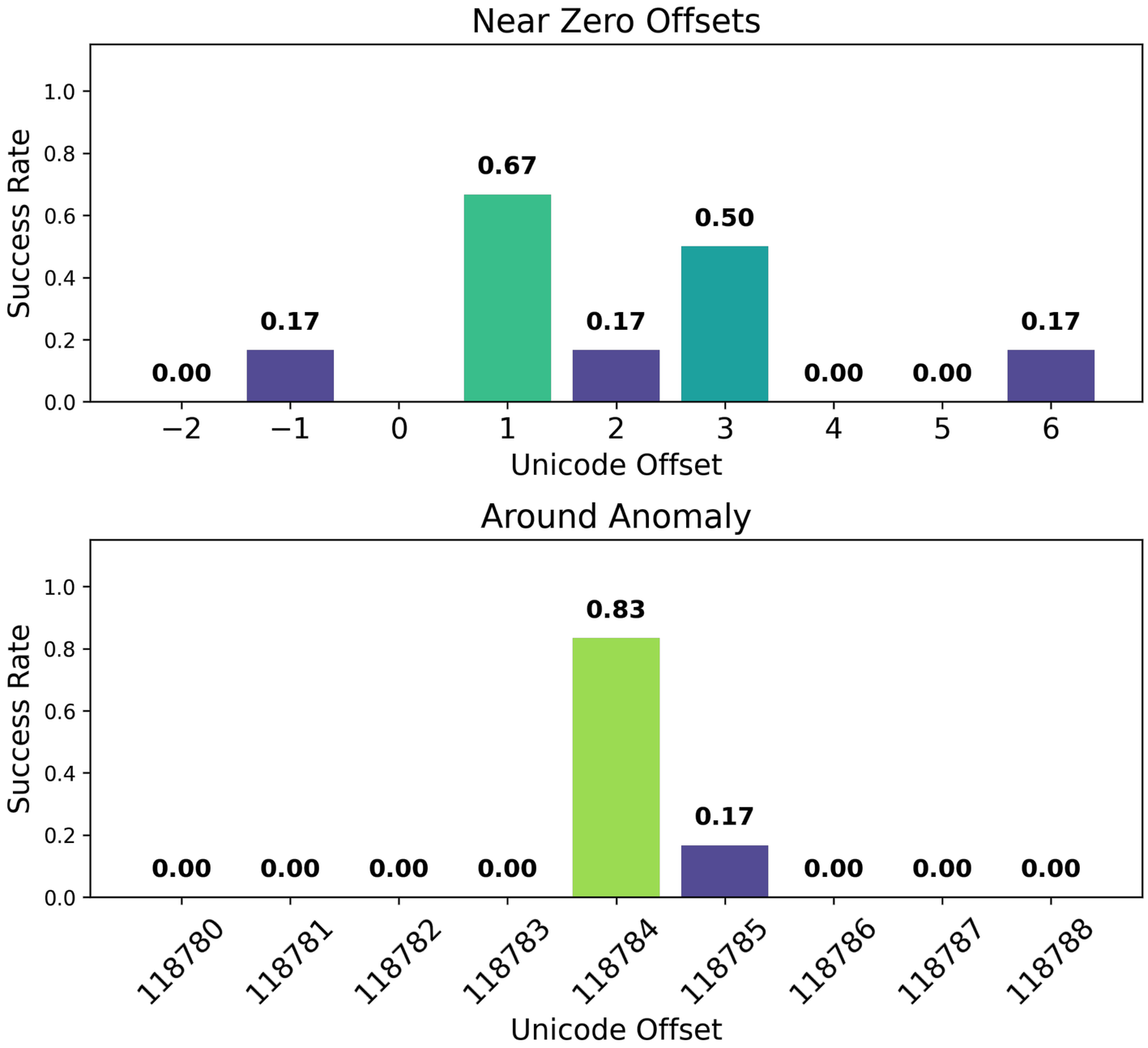
新聞，但總沒有想到自己可以照樣做，現在看見趙司晨也如此，才有了學樣的意思，定下實行的決心。他用一支竹筷將辮子盤在頭頂

當初很不快，後來便很不平。他近來很容易鬧脾氣了；其實他的生活，倒也並不比造反之前反艱難，人見他也客氣，店鋪也不說要現

竹筷。阿Ｑ萬料不到他也敢這樣做， 自己也決不准他這樣做！小Ｄ是什麼東西呢？他很想即刻揪住他，拗斷他的竹筷，放下他的辮子
唾沫道「呸！」

本也想靠¨寄存箱子的淵源，親身去拜訪舉人老爺的，但因為有剪辮的危險，所以也就中止了。他寫了一封「黃傘格」〔４７〕的信
一塊銀桃子掛在大襟上了；未莊人都驚服，說這是柿油黨的頂子〔４８〕，抵得一個翰林〔４９〕；趙太爺因此也驟然大闊，遠過於
，一聽得這銀桃子的傳說，他立即悟出自己之所以冷落的原因了：要革命，單說投降，是不行的；盤上辮子，也不行的；第一¨仍然要和
除卻趕緊去和假洋鬼子商量之外，再沒有別的道路了。

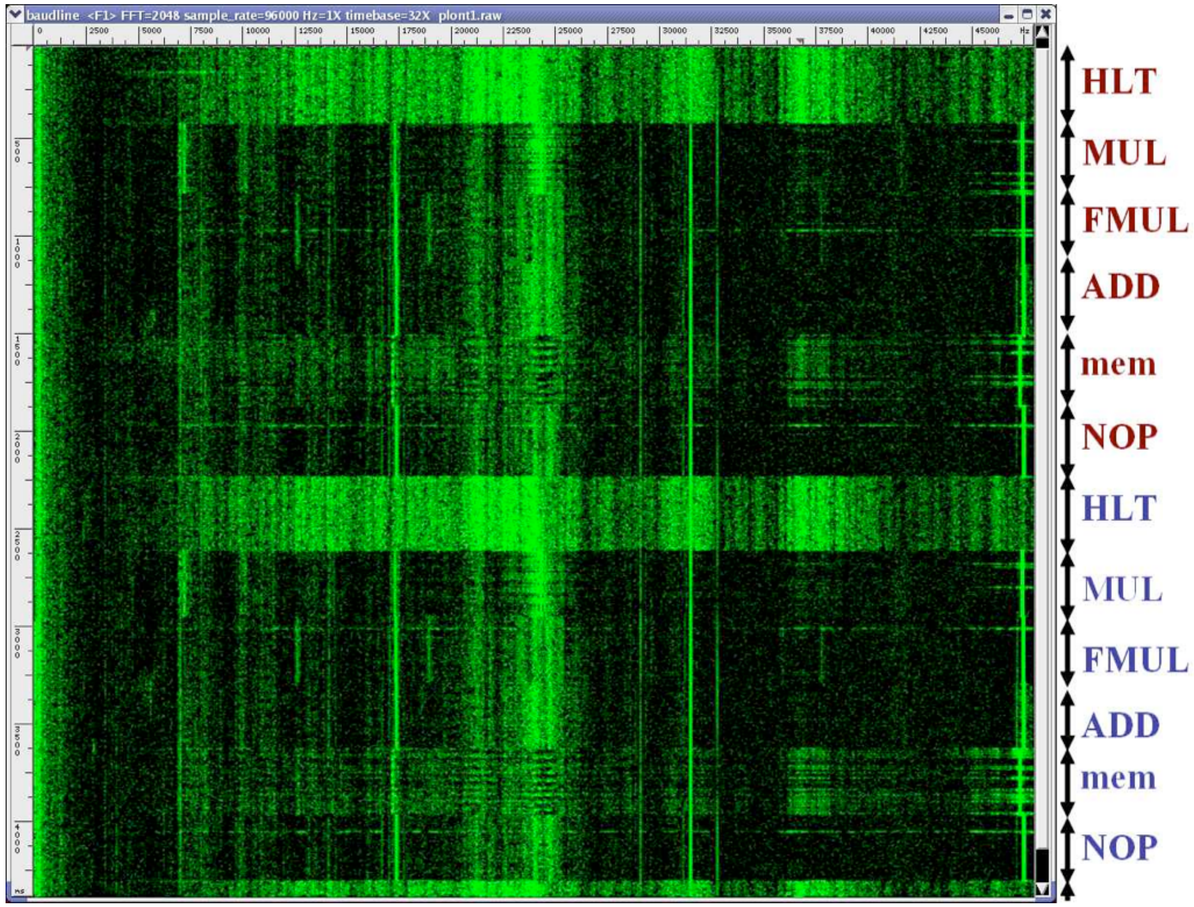
Unicode Offset Decoding Success Rates - gpt-4o



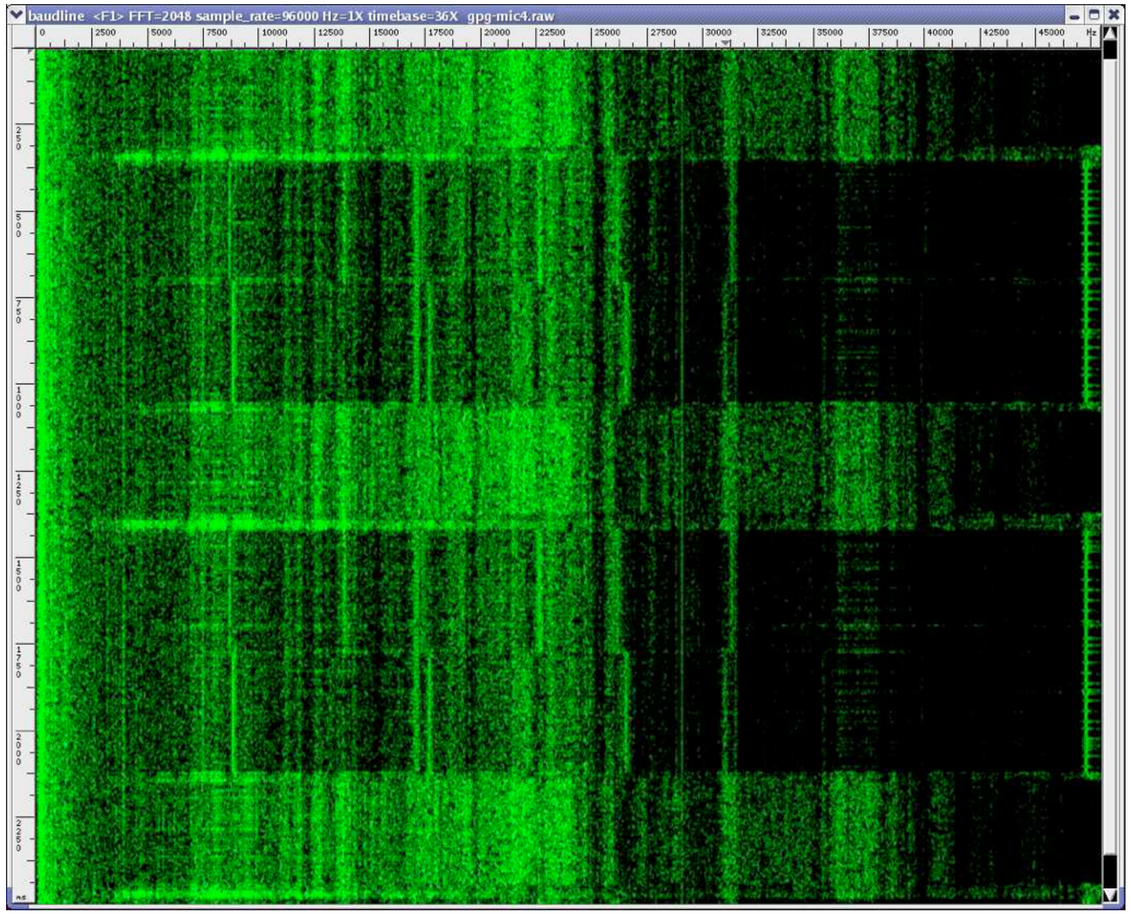
Review

- Side channels (discussed last week)
- Finishing up one topic
- Going briefly over Spectre + Meltdown

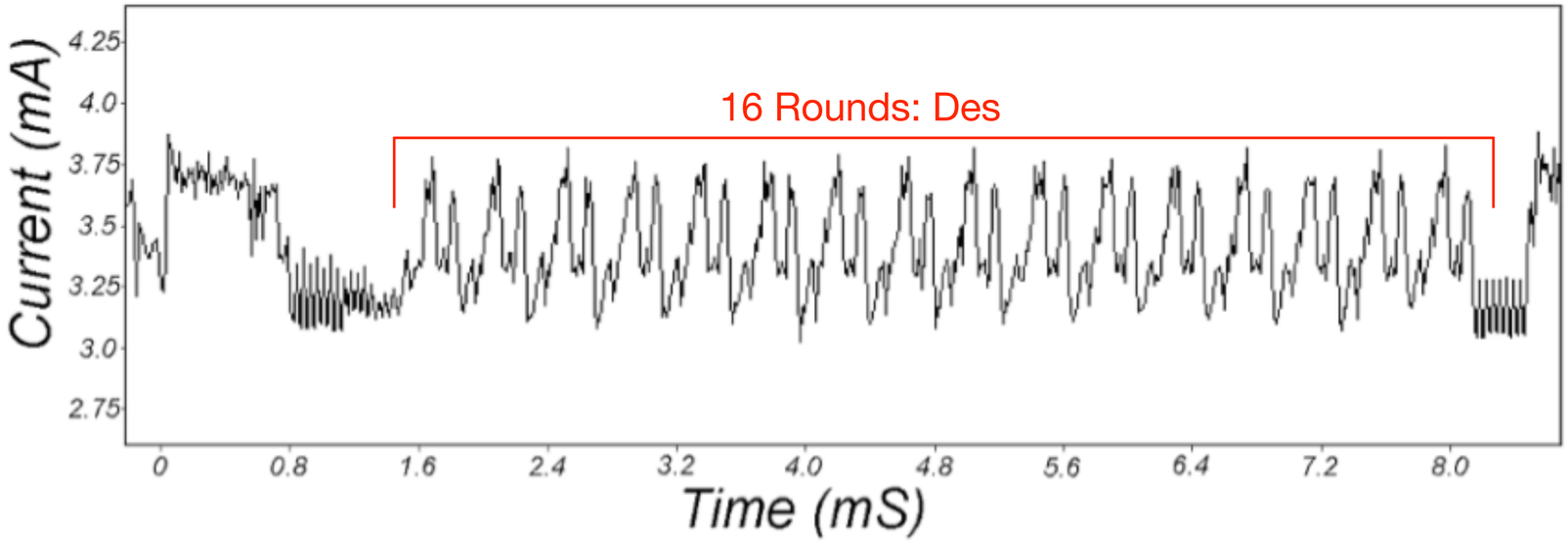
Acoustic Side Channels



Acoustic Side-Channels



Reverse Engineering



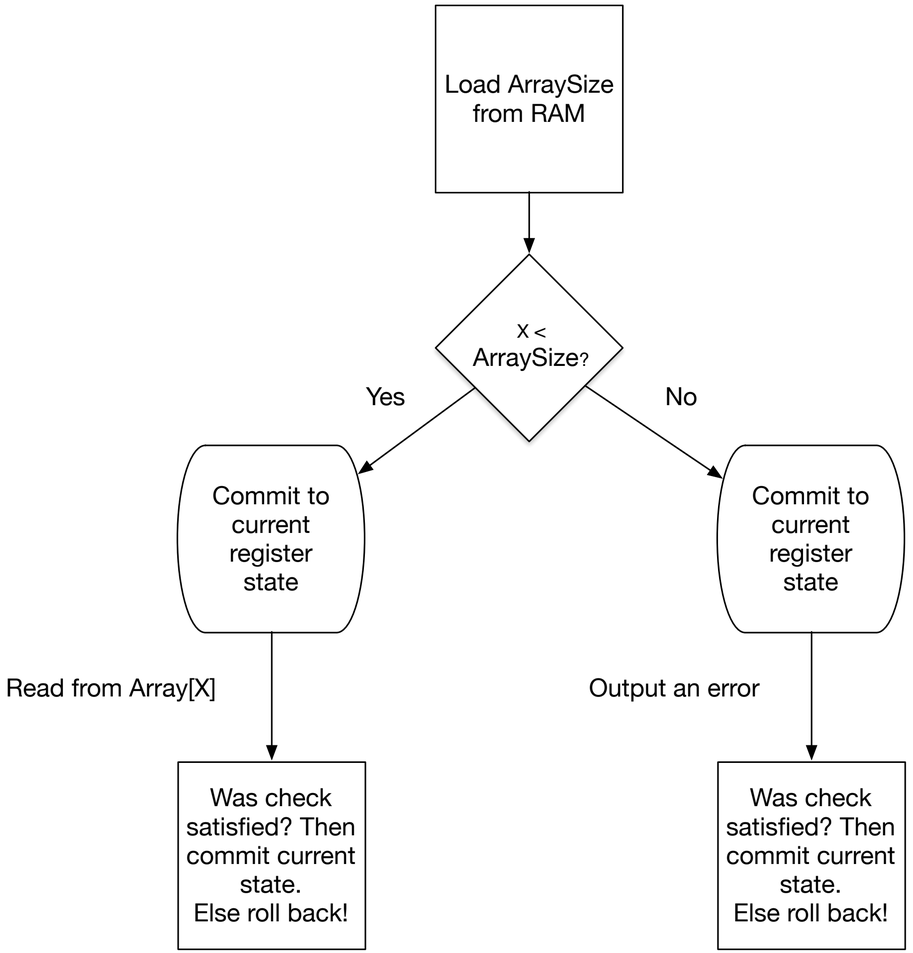
Speculative Execution

- Programs often branch, based on values drawn from RAM
  - RAM accesses are slow
  - Modern processors could wait, but this would waste cycles
  - So instead, they “guess” the right value and perform speculative execution

Speculative Execution

```
If (X < ArraySize) {
  y = array2[array[X] * 4096];
} else {
  // output an error
}
```

Speculative Execution

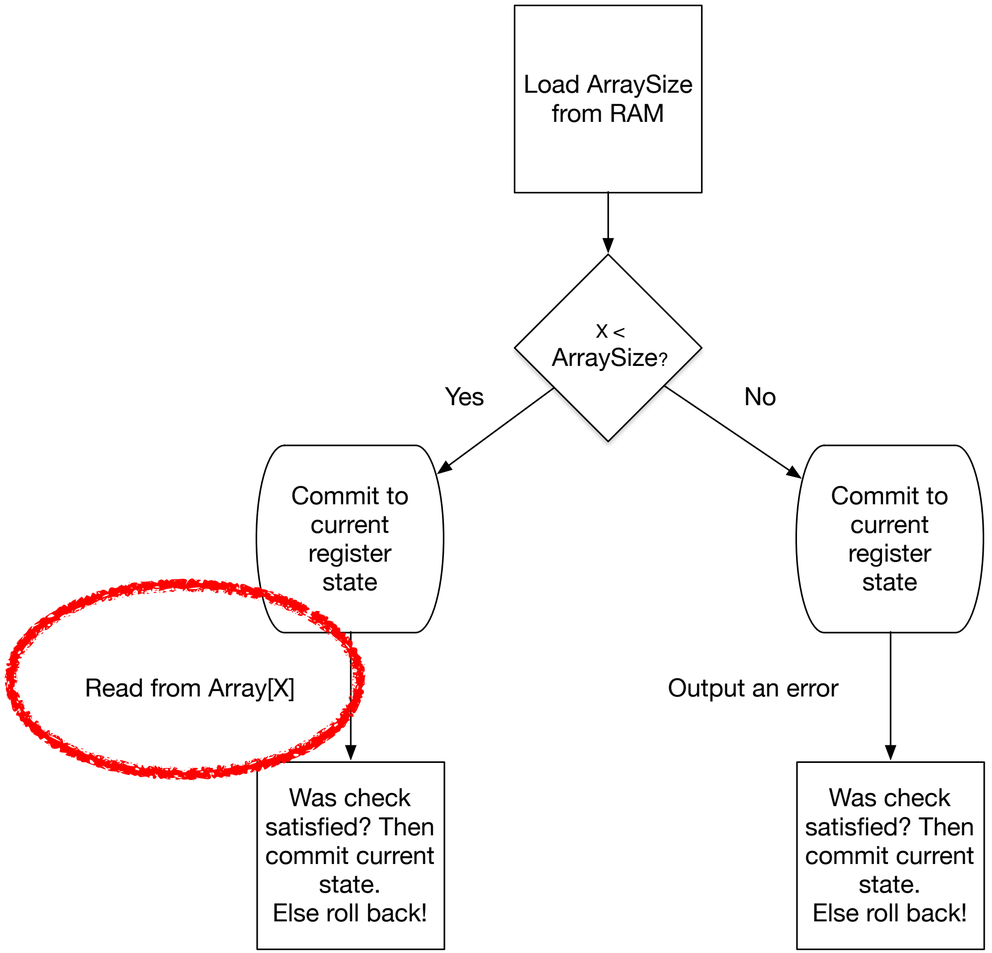


Speculative Execution

```
If (X < ArraySize) {
  y = array2[array[X] * 4096];
} else {
  // output an error
}
```

* X is attacker controlled
* ArraySize is in RAM
* array[X] is in *cache*
* Processor must speculate on (X < ArraySize) without knowing that value.

Speculative Execution



Spectre

- In theory, rollback “eliminates”
  the results of a speculative execution
  that was not supposed to run
- In practice, however, processors are complicated…
- Calculation involving invalid X may affect cache state
  - Specifically, cache lines associated with array2[X*4096] will be affected by branch

Spectre

- 1. Train branch predictor on many valid
  2. “Evict” ArraySize and array2 from cache
  3. Pass in an attacker controlled X
     that is well past the array, where
     array[X] is in cache
  4. Examine which cache lines have been
     affected

```
   If (X < ArraySize) {
y = array2[array[X] * 4096];
         } else {
     // output an error
            }
```

Who chooses the branch?

- This is done by a “branch predictor”
- The predictor has memory, and remembers which branch it chose in a program it has run before
- It can be “trained” by an attacker to prefer one branch, regardless of the data it gets

Spectre (variant 2)

- Speculative execution can do more than "if/then" branches
- For example, if you jump to an address, and the address isn't in cache, the predictor will try to "guess" which code should run
- It does this based on past experience
- This experience may have been derived from the execution of other processes

Spectre (variant 2)

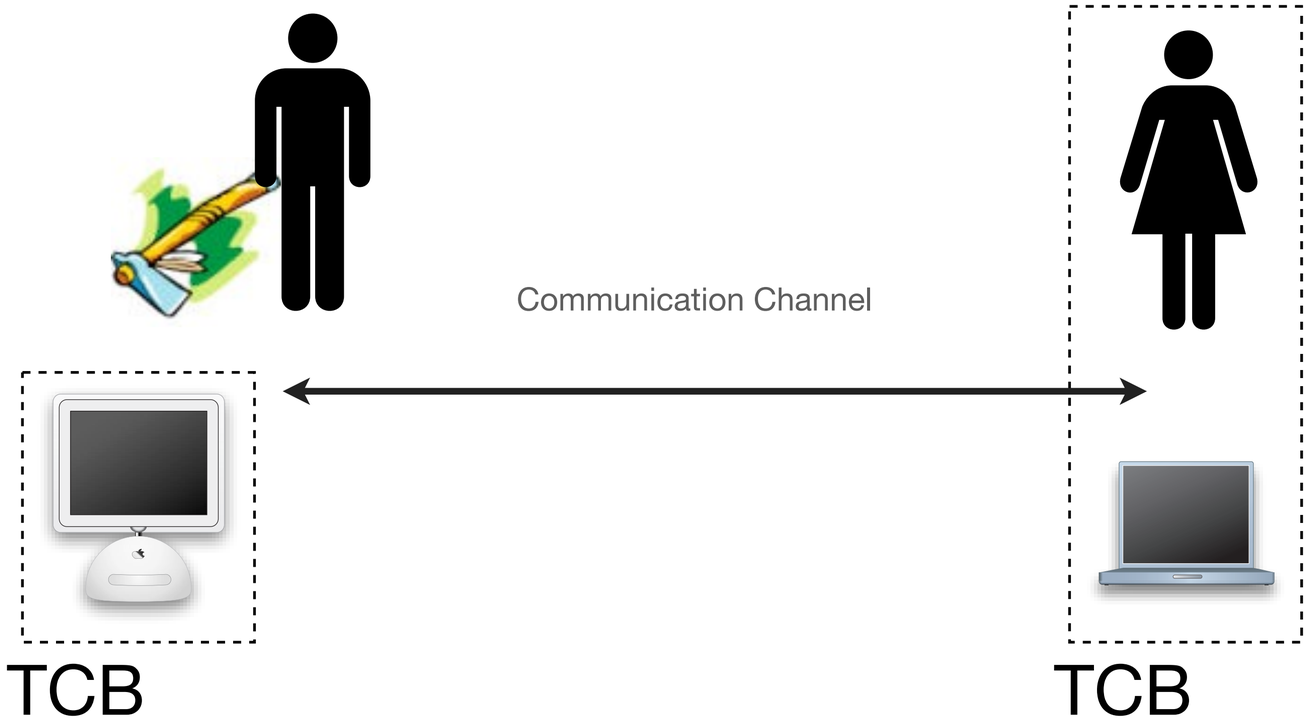
- The actions of one program/process influence the branch predictor's behavior in another
- If Process A (attacker controlled) repeatedly indirect-branches to memory address X,
  then Process B (victim) encounters a branch with X, the predictor will prefer that branch.
- So attacker can "train" the predictor in Process A to affect Process B.

Spectre (variant 2)

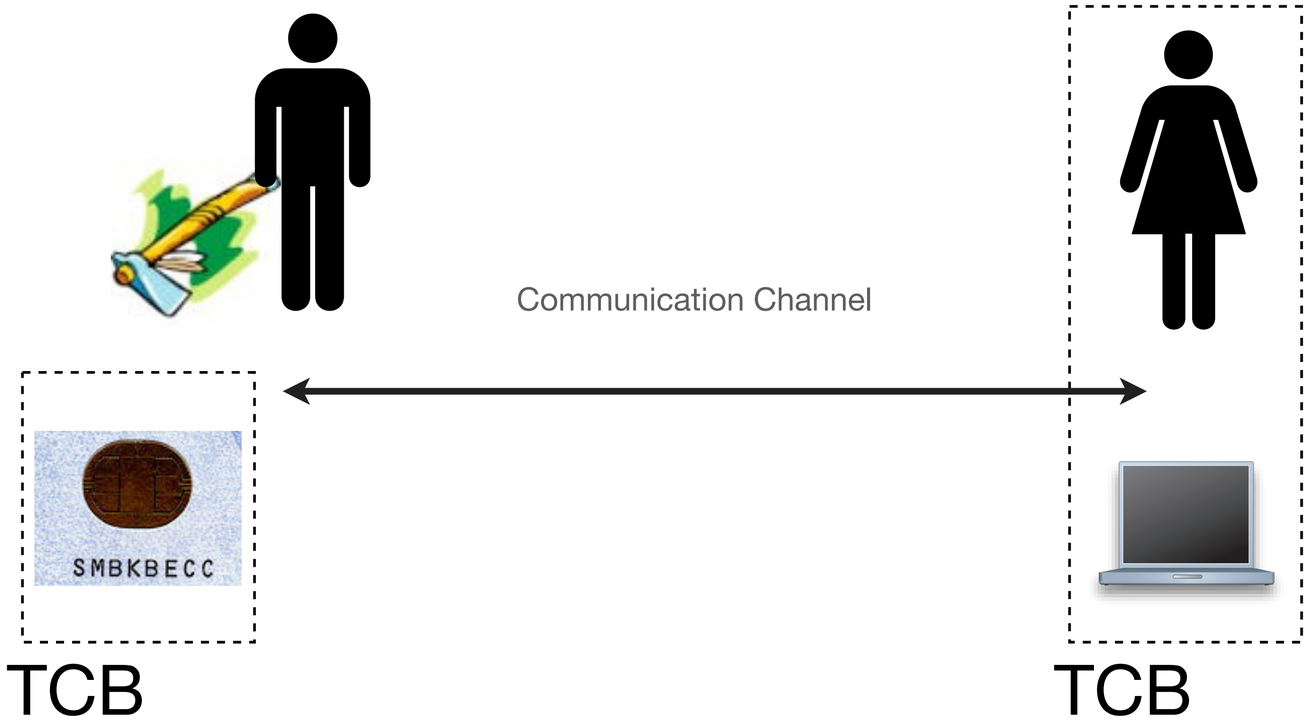
- Identify a gadget G in the victim program
  - This is a chunk of code that e.g., accesses some data referred to by a specific register, uses this to make a second read
- Attacker process “trains” the predictor so that it will jump to the address of the Gadget in the attacker process
  - (This means the predictor may do the same thing in the victim process as well.)
- Attacker sends attacker-controlled to victim program
-

Intel SGX

Today



Today



Physical Security

- Devices in a hostile environment
  - Military cryptoprocessors
  - Financial services

•ATM machines

•Credit-card smartchips

- Digital Rights Management

•Trusted Computing

Threat Models

- Various levels of sophistication
  - Class 1: Clever Outsiders
    Moderate knowledge of system, will usually attack via a known weakness
  - Class 2: Knowledgeable Insiders
    Specialized technical education & experience
    Access to the system & tools to exploit it
  - Class 3: Funded Organizations
    Team of attackers backed by significant funding,
    access to experts and Class 2 attackers

FIPS Levels

- FIPS 140-2: Cryptographic Modules
  - FIPS 140, Level 1

•Software only, no physical security

  - FIPS 140, Level 2

•Tamper evidence or pick-resistant locks

  - FIPS 140, Level 3

•Tamper resistance

  - FIPS 140, Level 4

•Strong physical security around device

System Examples

- Classical ATM machine:
  - Armor, temperature sensors, tilt sensors
  - Attacker has to drill through the shell, defeat any tamper sensors, avoid setting off alarms
  - Might lose $$, but can often protect cryptographic secrets

System Examples

- Modern ATM machine:
  - No armor, limited physical security
  - Owner/operator might be untrustworthy
  - Limited $$ amounts, still worry about loss of key material
  - Interface based on commodity OS (Windows CE?)

System Examples

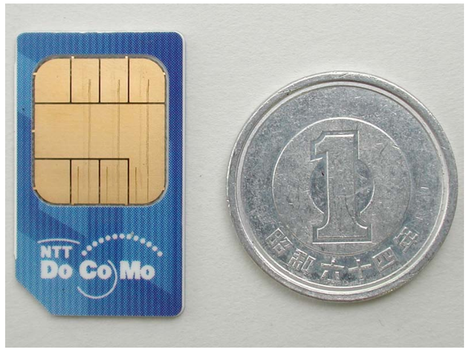
- Back-end server:
  - Strong physical security around server room
  - Performance is critical
  - Insider attacks a major concern
  - Large potential $$ losses if key material com
  - Linux/Windows/Legacy

System Examples

- Client-side smartcard:
  - User/owner is potentially hostile
  - Worse if card is stolen
  - Attacker-su ver

Two approaches

- Tamper-evidence
  - Make tampering obvious
  - Allow for recovery/renewability
    (e.g., re-generate cryptographic keys)
- Tamper-resistance:
  - Keep attackers out
  - Wipe cryptographic key material



A General Approach

- Minimize the size of the “T” in our TCB
  - I.e., putting an ATM in a safe is expensive
  - Anchor trust in this area, build a secure system from there
  - Might involve untrusted storage/RAM/ROM/ processing, all connected to one trusted component

A General Approach

- Might also be necessary for practical reasons:
  - Support multiple “untrusted” manufacturers
  - E.g., SIM chips
  - E.g., CableCARD

Protecting Cryptoprocessors

- Metal
  - Can be cut/drilled
- Epoxy
  - Can be scraped off, penetrated with a logic analyzer probe

IBM 4758

IBM 4758

- High-security Cryptoprocessor
  - RAM, CPU, cryptographic circuitry
  - Dedicated SRAM for key memory
  - Battery powered
  - All wrapped in:

•Aluminum EM shielding

•Tamper-sensing mesh

•Potting material

IBM 4758

- Tamper sensing mesh:
  - Thousands of small wires wrapped around device
  - Interrupting any circuit (i.e., drilling) wipes the contents of key SRAM

•v1: copper wires

•v2: printed circuits

IBM 4758

- Memory remanence attacks:
  - Key “burn in”
  - After storing keys for too long, they may become permanent default states of RAM
  - NSA Forest Green Book has guidelines for this
  - Solution: “RAM savers”

IBM 4758

- Memory remanence attacks:
  - Attacker might try to freeze the device
  - Thus keys would survive wipe
  - Bursts of X-Rays can “lock” RAM in
  - Protections: temperature sensor, radiation sen

•Gets too cold, keys go away

•Though what if we ship UPS!

Capstone/Clipper

- Proposed national voice encryption device
  - Designed as a compromise between need for strong crypto, and US's need to eavesdrop
  - Contained a secret cipher (Skipjack) w/80-bit key
  - Tamper-resistance:

•Difficult to extract embedded secret keys

•Difficult to R.E. Skipjack design

•Difficult to alter operation

  - Currently used in Fortezza card (US gov't SECRET)

Capstone/Clipper

- Threat model & design considerations
  - Extremely hostile environment
  - Range of well-funded adversaries (probably non-military)
  - Protecting secrets & design & operation
  - Very small device

Clipper/Capstone

- Tamper-resistance:
  - Extremely sophisticated
  - Metal/epoxy top
  - Vialink Read-Only Memory (VROM)

•Bits set by blowing antifuses using electrical charges

Clipper/Capstone

- Attacking Clipper & QuickLogic:
  - Remove upper metal layer (difficult)
  - Use an electron microscope to read VROM (expensive & difficult, requires extra analysis)
  - Monitor circuit while chip is in use (more promising)

Capstone/Clipper

- Operation:
  - Each ciphertext accompanied by LEAF (Law Enforcement Access Field)
  - Essentially, key escrow under gov’t key
  - LEAF contains a “checksum” (MAC) to prevent tampering/removal
- Break:
  - Matt Blaze - found that LEAF bound to message w/ 16-bit checksum

Smartcards

- Very widely used
  - Contact/Contactless varieties
  - Small microprocessor/RAM/Serial bus
  - Became major targets of attack due to:

•Satellite TV

•GSM phones

•Lately: Payment Cards

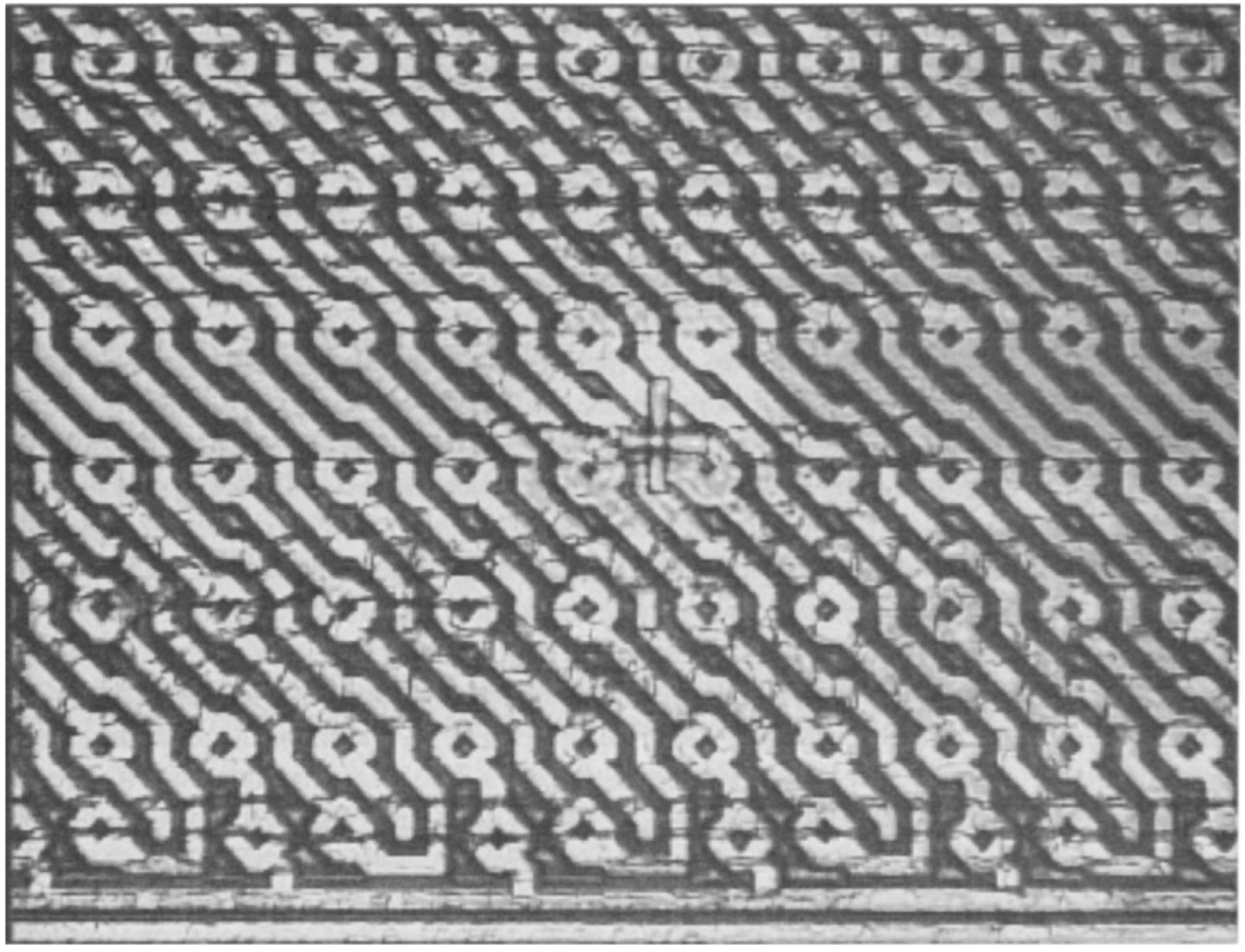
Attacking Smartcards

- Protocol-level attacks
  - Intercept messages, control access
- Side-channel attacks
  - Attacker controls the power source
  - DPA countermeasures introduces in 1990s

Attacking Smartcards

- Attacks on voltage supply
  - Memory read/write attacks
  - Fault injection
- Attacks on hardware
  - Take it apart, use an electron microscope
  - Same countermeasures, though:

•protective mesh, potting



Tamper-evidence

- Seals/locks
  - Make sure you can detect and renew security after an attack occurs
- Can even be implemented in software (under certain assumptions)

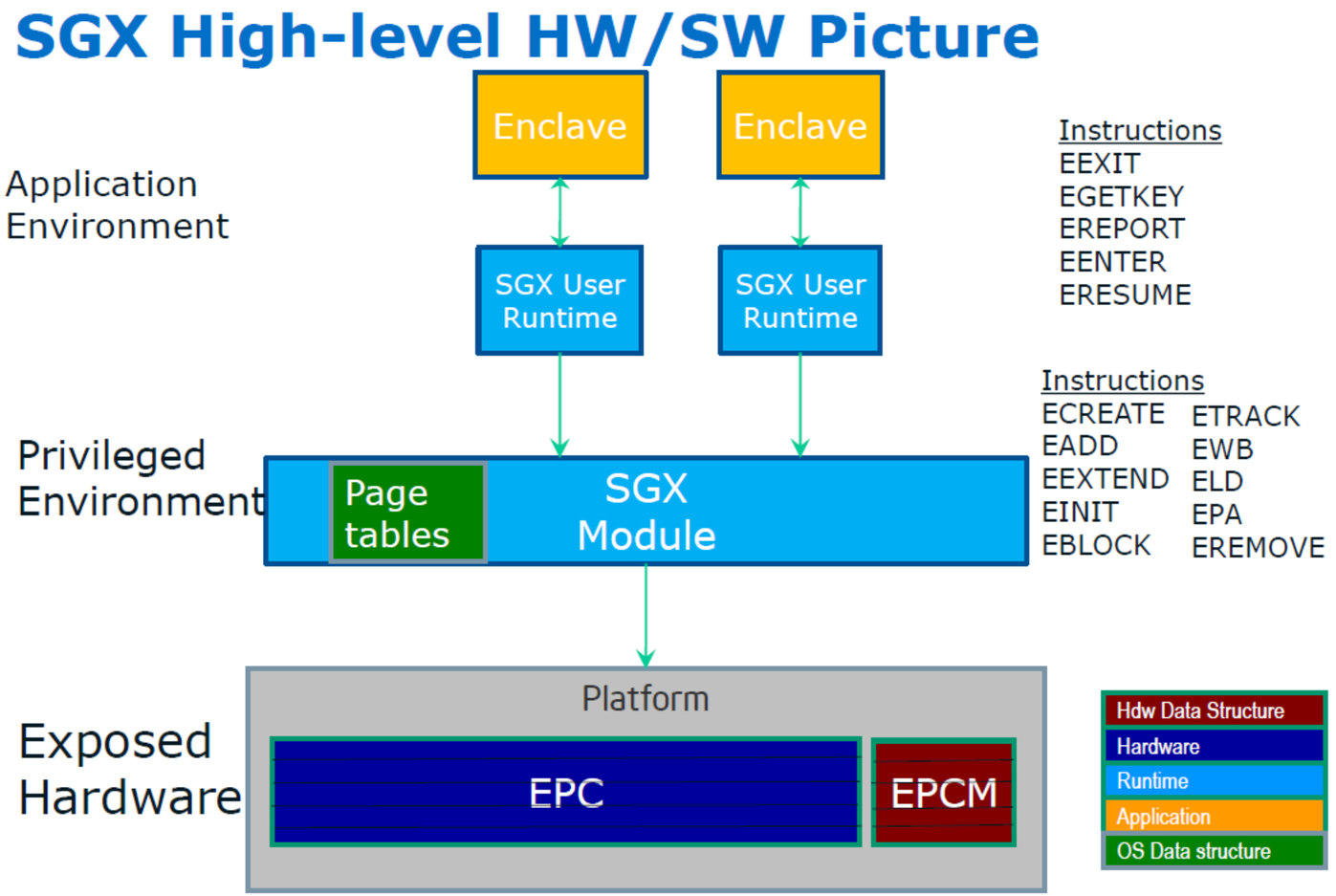
Trusted Computing

- Effort to put a tamper-resistant security processor into every PC
  - Useful to verify software integrity
  - Key management for access control & DRM
  - Minimize co-processor footprint by bootstrapping secure software
- Newest incarnation: Intel SGX

Trusted Computing

- Effort to put a tamper-resistant security processor into every PC
  - Useful to verify software integrity
  - Key management for access control & DRM
  - Minimize co-processor footprint by bootstrapping secure software
- Newest incarnation: Intel SGX

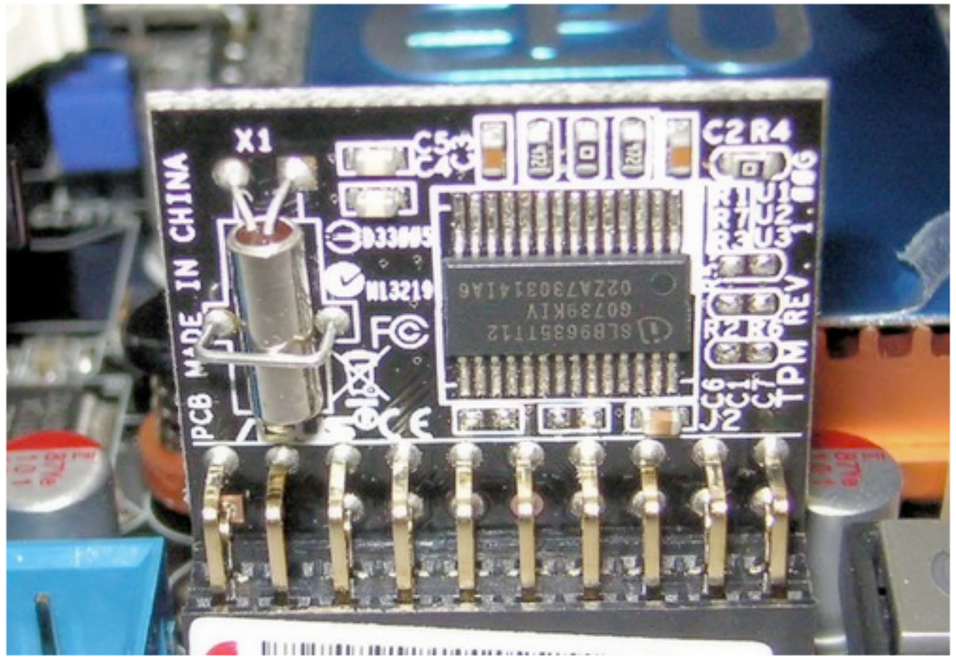
SGX


SGX High-level HW/SW Picture
Application Environment
Privileged Environment
Exposed Hardware
Enclave
Enclave
SGX User Runtime
SGX User Runtime
Page tables
SGX Module
Platform
EPC
EPCM
Instructions
EEXIT
EGETKEY
EREPORT
EENTER
ERESUME
Instructions
ECREATE
EADD
EEXTEND
EINIT
EBLOCK
ETRACK
EWB
ELD
EPA
EREMOVE
Hdw Data Structure
Hardware
Runtime
Application
OS Data structure

FILED UNDER *Desktops*, *Laptops*

Christopher Tarnovsky hacks Infineon's 'unhackable' chip, we prepare for false-advertising litigation

By Tim Stevens posted Feb 12th 2010 10:31AM



As it turns out, Infineon may have been a little bit... *optimistic* when it said its SLE66 CL PE was "unhackable" -- but only a little. The company should have put an asterisk next to the word, pointing to a disclaimer indicating something to the effect of: "Unless you have an electron microscope, small conductive needles to intercept the chip's internal circuitry, and the acid necessary to expose it." Those are some of the tools available to researcher Christopher Tarnovsky, who perpetrated the hack and presented his findings at the Black Hat DC Conference earlier this month. Initially, Infineon claimed what he'd done was impossible, but now has taken a step back and said "the risk is manageable, and you are just attacking one computer." We would tend to agree in this case, but Tarnovsky still deserves serious respect for this one. Nice work, Big Gun.

Trusted Computing

- Implications
  - Same chip used in the XBox 360
    (allegedly, Tarnovsky offered $100k to hack it)
  - Illustrates the most important lesson
    of tamper resistant systems:

•Individual devices can and will be hacked

•Must ensure that single-device hack
(or easily replicable attack)
does not lead to system-wide compromise!