Password hashing at scale

(for Internet companies with millions of users)

Solar Designer <solar@openwall.com>

@solardiz @Openwall

http://www.openwall.com

Yet another Conference

October 1, 2012

Moscow

Historical background

HTTP://OPENWALL.COM/PASS

Concepts to be familiar with:

- Password hashing
- Key derivation function
- Salting
- Password stretching
  - bcrypt, PBKDF2
- Memory-hard functions
  - scrypt

Threat models

- Offline attacks
  - Protected local parameter
  - Decent hash type
  - Password stretching
  - Random per-account salts

    With targeted attacks, salts are of less help, yet they should be used in those cases as well
  - Strict password policy
- Password reuse

  (across multiple sites)
- Online attacks
  - Password policy
  - Per-source rate limiting
  - Multi-factor authentication
  - Behavior analysis

    Akin to a spam filter
  - User-targeted attacks

    Phishing, trojans, client vulnerability exploits
  - Network-based attacks

    DNS, routing, MITM, sniffing
  - Server vulnerability exploits

Local parameter

- Must contain sufficient entropy
  - way beyond a typical password or even passphrase
- Hashes are not crackable offline without knowledge of the local parameter
- However, if the local parameter is stored right on the authentication server or in the password database, then it is likely to be stolen/leaked along with hashes
- Problem: migration of hashes between systems
  - Solution: embed a "local parameter ID" in the hash encodings, support multiple local parameters at once

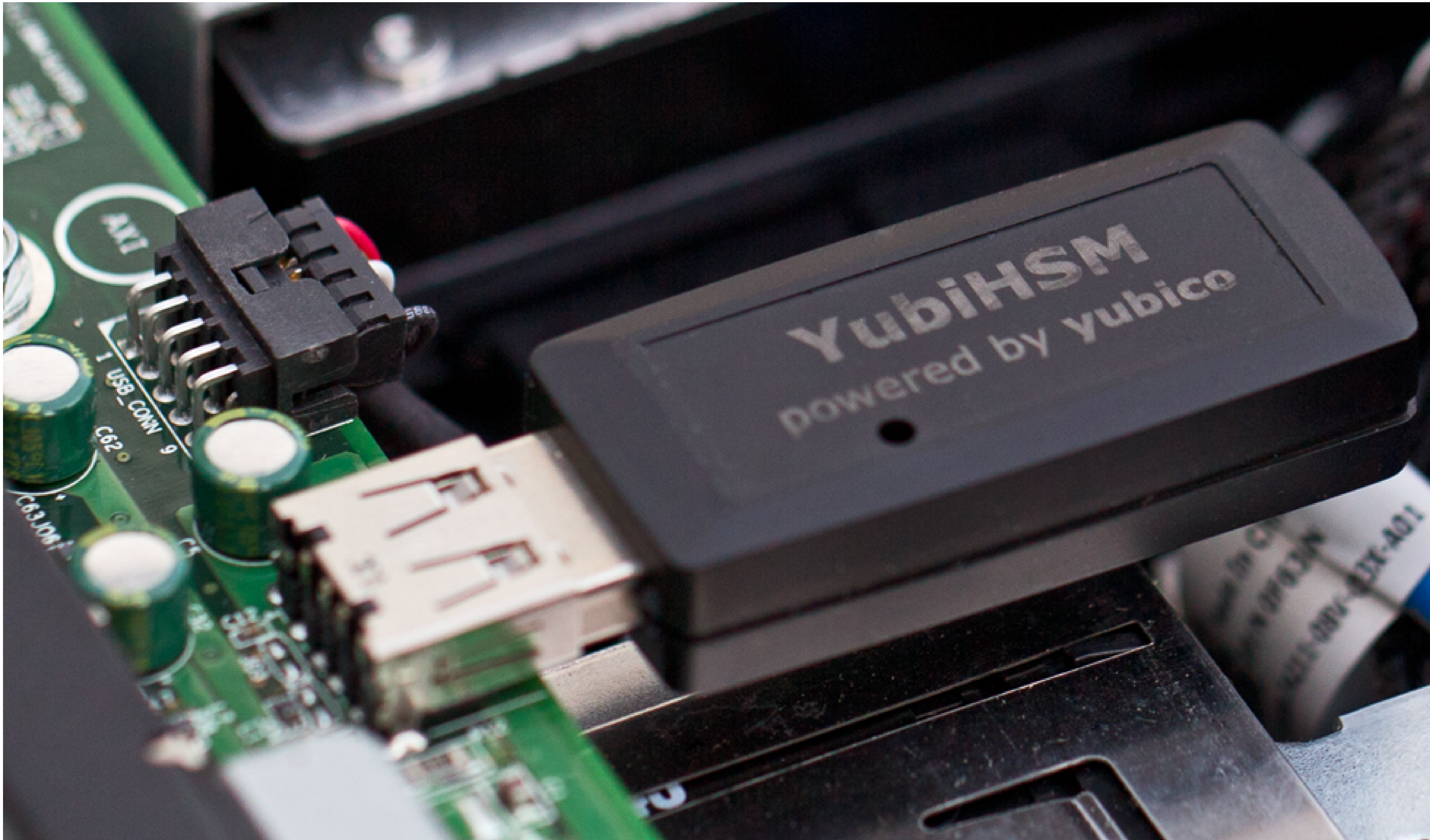
Unreadable local parameter

- When password hashing is at least partially implemented in a dedicated device (e.g., in a hardware security module or a dedicated server), it becomes possible to embed a local parameter in the device
- If the local parameter is unreadable by the host system (e.g., by a server doing password authentication), this buys us an extra layer of security
  - Need to have a backup copy - e.g., a cluster of multiple HSMs or/and a piece of paper in CEO's safe

Network structure (logical)

- Authentication servers
  - Receive usernames and passwords, reply with yes/no or a token
  - Optionally perform the costly portion of password hashing
  - Access the database, talk to password hashing HSMs or servers for the portion involving the local parameter
- Password hashing HSMs or servers
  - Are accessible from the authentication servers only
  - Receive partially computed hashes or passwords to hash, return computed hashes
- Other servers needing user authentication
  - Talk to authentication servers or/and accept tokens

YubiHSM - a USB dongle for servers



YubiHSM in a server's internal USB port. Photo (c) Yubico, reproduced under the fair use doctrine.

Local parameter in YubiHSM

YubiHSM provides several suitable functions.

If we use HMAC-SHA-1:

- Key is the local parameter
- "Key handle" is its ID
- "Data" is output of a KDF
- HMAC is password hash

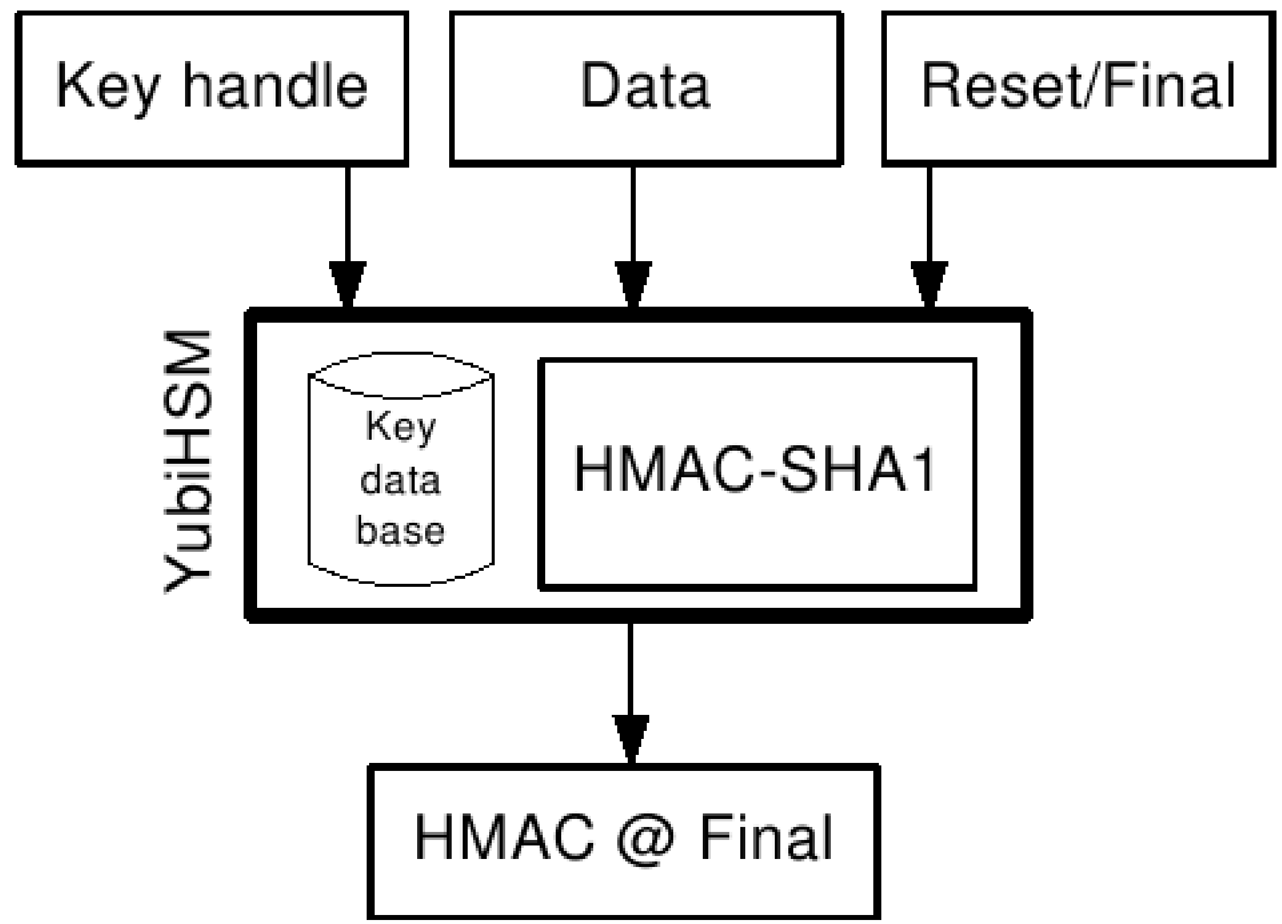


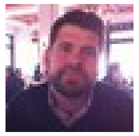
Diagram (c) Yubico, reproduced under the fair use doctrine

YubiHSM pros

- Similar purpose, thus the right threat model
- Per key permission flags
- No custom OS kernel level driver required (USB CDC)
- Well-documented APIs, sample code
- Low cost ($500; other HSMs may be $10k to 20k EUR)
  - You need at least two for redundancy
- Independent formal analysis of the Yubikey protocol
  - "YubiSecure? Formal Security Analysis Results for the Yubikey and YubiHSM" by Robert Künnemann and Graham Steel, INRIA
    - Assumes "that the implementation is correct with respect to the documentation"
    - Found an oversight, which Yubico has since released a security advisory on

YubiHSM cons

- No independent whitebox audits
- No independent blackbox audits
  - probing for implementation issues



Matthew Green
@matthew_d_green

.@agl__ @ioerror HSMs can and should be audited. Obviously this isn't trivial, but it's critical if you're going to use one.

- USB CDC is slow, up to ~500 requests per second sustained throughput
- Serial interface for block-oriented data is risky
  - "careful design is required not to lose synchronization in a serial byte stream" (Yubico)
- Not tamper-resistant (physical attacks are outside of the threat model)

Issues with HSMs in general

- Purpose and threat model are not always suitable
  - Crypto acceleration or/and security
    - At least symmetric crypto is often not faster than optimized code on CPU anyway
  - Attacks from compromised host or/and physical
- Potential vulnerabilities
  - Firmware bugs, design errors, side-channels
    - Attack surface (too many features, each being a risk - can disable or not?)
  - No known whitebox audits, source code not available for review
- Interfaces (physical, driver, API) and their reliability
- Cost is often significant
  - Especially given that multiple HSMs need to be installed

Speed of offline attacks (with salts)

Assumptions:

- Unique per-user salts
- Non-targeted attack
  - Accounts are of equal value
  - No password strength hint

It is tough to limit offline attack speed to 1000/s (by password stretching). Obviously, if we need to handle more than 1000 requests/s ourselves, an attacker with the same resources will also be able to try at least as many.

Guesses / second	Users	Daily guesses / user
1,000	1	86,400,000
1,000	1,000	86,400
1,000	1,000,000	86
1,000	100,000,000	1
1,000,000,000	1	86,400,000,000,000
1,000,000,000	1,000	86,400,000,000
1,000,000,000	1,000,000	86,400,000
1,000,000,000	100,000,000	864,000

1 billion/s is a conservative GPU attack speed estimate for hashes without password stretching. In practice, multi-billion speeds are often achieved.

Password stretching

- 100 ms is commonly suggested, but is it affordable?
- Maybe not for every use case, but even if so stretching must be used anyway - just at a lower setting
  - Even if we merely slow down an offline attacker from billions/s to millions/s, this is worthwhile - and we'll do more than that
- 1 ms ought to be affordable for anybody?
  - Allows for up to 1000 requests/s/core, theoretically up to 86 million requests/day/core - but need to leave room for spikes
  - If average is 10x lower than the worst spike we need to support, a 12-core server will handle up to ~100 million requests/day
  - Need more? You surely can afford more servers (at least N+1)

Hash type matters

- Attackers might not use the same kind of hardware and software as ours
  - They might use a more suitable, attack-optimized setup
  - They might have a preference to use whatever they readily have (e.g., existing GPU rigs, botnets)

- A good hash type to use is:
  - friendly to our hardware
  - unfriendly to hardware that we do not anticipate to use
  - efficiently implemented for defense
  - does not allow for much additional optimization for attack

What's wrong with PBKDF2

As commonly used with HMAC-SHA-*

- No parallelism - slows down defender, but not attacker
  - When implemented on modern CPUs for defensive use, only a relatively small portion of resources available in one CPU core is used (can't use SIMD, low instructions per cycle)
- Almost no memory needs - defender's RAM is not put to use, attacker does not need to provide RAM
- GPU friendly
  - More so with SHA-1 than with SHA-512, though

    SHA-512 uses 64-bit words, which helps CPUs and hurts current GPUs

What's wrong with bcrypt

- No parallelism, 32-bit word size - slows down defender
  - Low instructions per cycle (attack is ~2x faster), can't use SIMD
  - Attacker's use of SIMD is also impacted, though - except on devices with scatter/gather addressing (or at least gather)
    - Intel MIC (2012, limited availability), AVX2 (2013, will be widespread?)
- Low memory needs (only 4 KB) - defender's off-chip RAM is not put to use (only L1 cache is), attacker does not need to provide DRAM
  - Yet due to bcrypt's memory access pattern this turns out to be (barely) enough to defeat GPUs so far (AMD Radeon HD 7970 is only about as fast as a CPU)

ASIC/FPGA attacks on modern hashes

- PBKDF2-HMAC-SHA-1
- PBKDF2-HMAC-SHA-256
- sha256crypt
- PBKDF2-HMAC-SHA-512
- sha512crypt
- bcrypt
- scrypt

Weaker

Stronger

It is a sound approach to consider attacks with ASICs, but in practice attacks with less flexible devices are also relevant

GPU attacks on modern hashes

- PBKDF2-HMAC-SHA-1
- PBKDF2-HMAC-SHA-256
- sha256crypt
- PBKDF2-HMAC-SHA-512
- sha512crypt
- scrypt at up to ~1 MB (misuse)
  - Litecoin at 128 KB is ~10x faster on GPU vs. CPU
- bcrypt (uses 4 KB)
- scrypt at multi-megabyte memory
- Revised scrypt with TMTO defeater

▲ Weaker

Stronger ▼

scrypt at low memory

- scrypt accesses memory in cache line sized chunks, which lets it use the memory bus efficiently
  - The attacker's cost is meant to be RAM itself, not bandwidth
- When scrypt is set to use only a small amount of memory (~1 MB or less), it is weaker than bcrypt at least as it relates to attacks on GPU
- At 128 KB, as demonstrated by scrypt's use in Litecoin, scrypt is ~10x faster on GPU than on CPU (whereas bcrypt is currently not faster on GPU than on CPU)
  - GPU cards' RAM bandwidth exceeds CPUs' L2 cache bandwidth

scrypt time-memory trade-off

- scrypt deliberately allows for a time-memory trade-off
  - "The design of scrypt puts a lower bound on the area-time product - you can use less memory and more CPU time, but the ratios stay within a constant factor of each other, so for the worst-case attacker (ASICs) the cost per password attempted stays the same"

    Colin Percival, crypt-dev mailing list posting, 2011
- Litecoin miners on GPU use this
- scrypt may be revised to defeat the trade-off
  - Pros: fewer pre-existing hardware devices (GPUs, etc.) are efficient in an attack
  - Cons: not official scrypt anymore, some defensive uses may be impacted as well (e.g., client-side hashing on mobile devices)

What's wrong with scrypt

- ~100 ms corresponds to 32 MB memory usage on current server hardware - we could afford more RAM
- At 1 ms, memory usage is so low that bcrypt is stronger
  - Experiment: in the reference implementation (the one with SSE2 intrinsics, running on x86-64), reduce the number of Salsa20 rounds from 8 to 2
  - Result: only ~2x increase in memory usage at the same duration
- Time-memory trade-off benefits attackers with GPUs
  - Can be fairly easily defeated, but then it's not official scrypt

A drawback of memory-hard KDFs

This applies to use of memory-hard KDFs for authentication in general, it is not specific to scrypt

- To use a lot of RAM fast, we need to get close to the full memory bandwidth, but this means poor scalability when many concurrent instances are run
- Thus, we have to choose between using more RAM per instance (and using CPUs' resources poorly when there are concurrent instances) and using CPU cores more fully (but at a lower RAM setting per instance)

Other memory-hard KDFs

Aside from some historical ones and bcrypt (which did not use more than a few KB), there appears to be only slothKdf (and its zen32 component) by Elias Yarrkov

- Only released as part of dhbitty program, may change
- Not peer-reviewed
- Uses the memory bus poorly (32-bit random accesses)
  - Provides advantage to attackers with custom hardware
- Bumps into the memory bandwidth
- A revision of zen32 with cache line sized accesses may use more RAM than scrypt (same duration)

GPUs for defense - tricky

- Need a lot of parallelism (thousands of work-items)
  - More than nearly-concurrent authentication attempts provide
- PBKDF2 and bcrypt lack parallelism
  - Yet PBKDF2-HMAC-SHA-1 is an excellent choice for GPU implementation if parallelism is added on top of it
- scrypt might be reasonable (large `p`, low memory)
- Involves many other trade-offs, challenges, risks
  - NVIDIA Tesla cards are suitable for servers, but slower for crypto than AMD's (so implement DJB's hash127 with floating-point?)
  - Lower reliability (than other components), driver bugs
  - Heat dissipation (use lower clock rate, duty cycle)

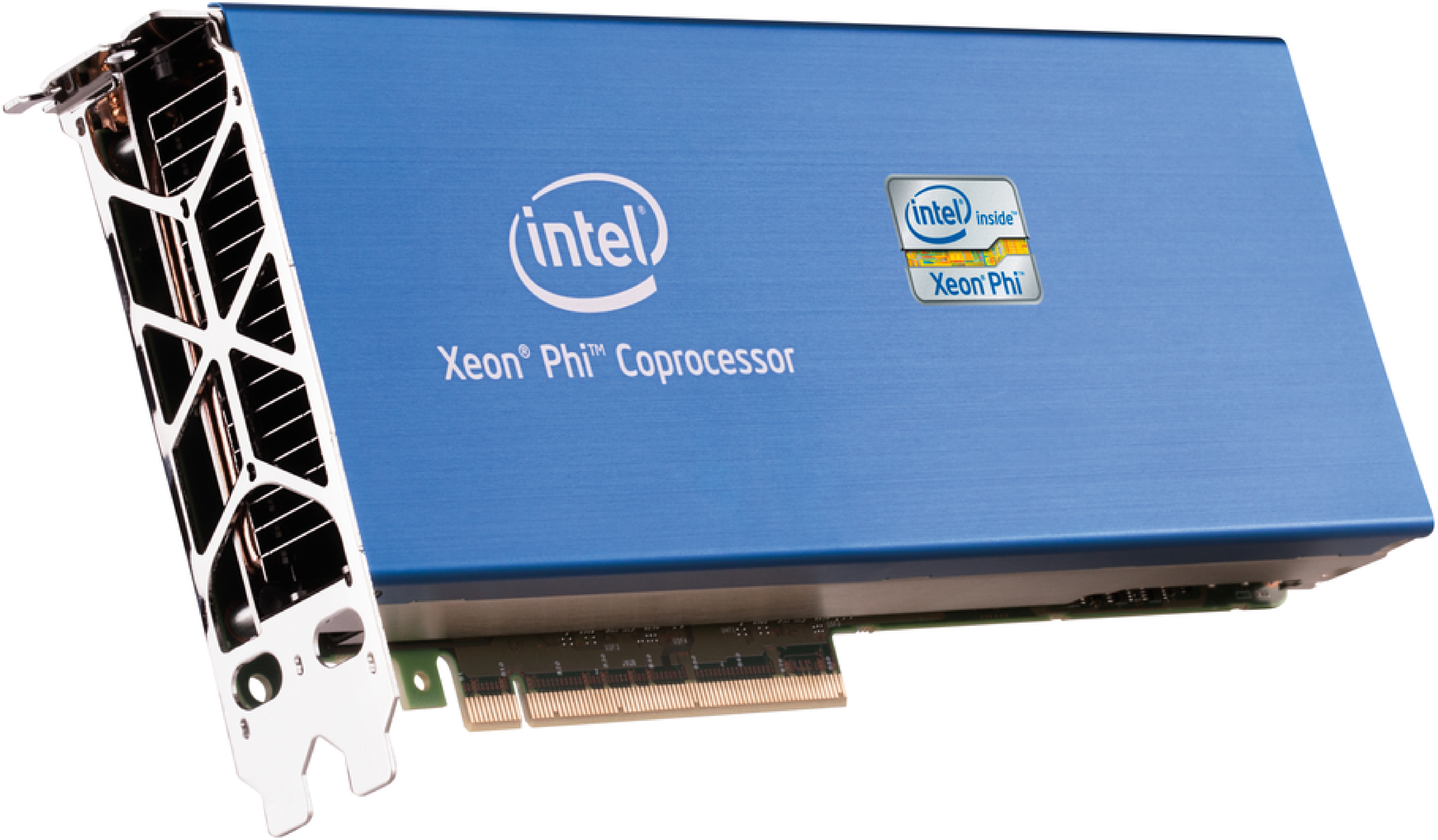
GPUs for defense - questionable

- What we gain by using a GPU (wisely)
  - A lot of computing logic is put to use (beyond a CPU's)
  - Can install up to 8 GPUs per machine easier/cheaper

    Many attackers also have GPUs and benefit from these same things, although most nodes in a botnet may be unsuitable
- What we lose by using only a GPU
  - Potential for unfriendliness to attackers with primarily GPUs
  - Memory requirements for attack, unless we manage to use each GPU card's global memory almost fully
- May combine use of GPUs with use of the host's RAM

  by two distinct components of the hashing method
  - This addresses the drawbacks above, but adds complexity

Intel Xeon Phi (Knights Corner)



Xeon Phi is a dual-slot PCIe card. Photo (c) Intel, released as part of press materials.

Xeon Phi in a nutshell

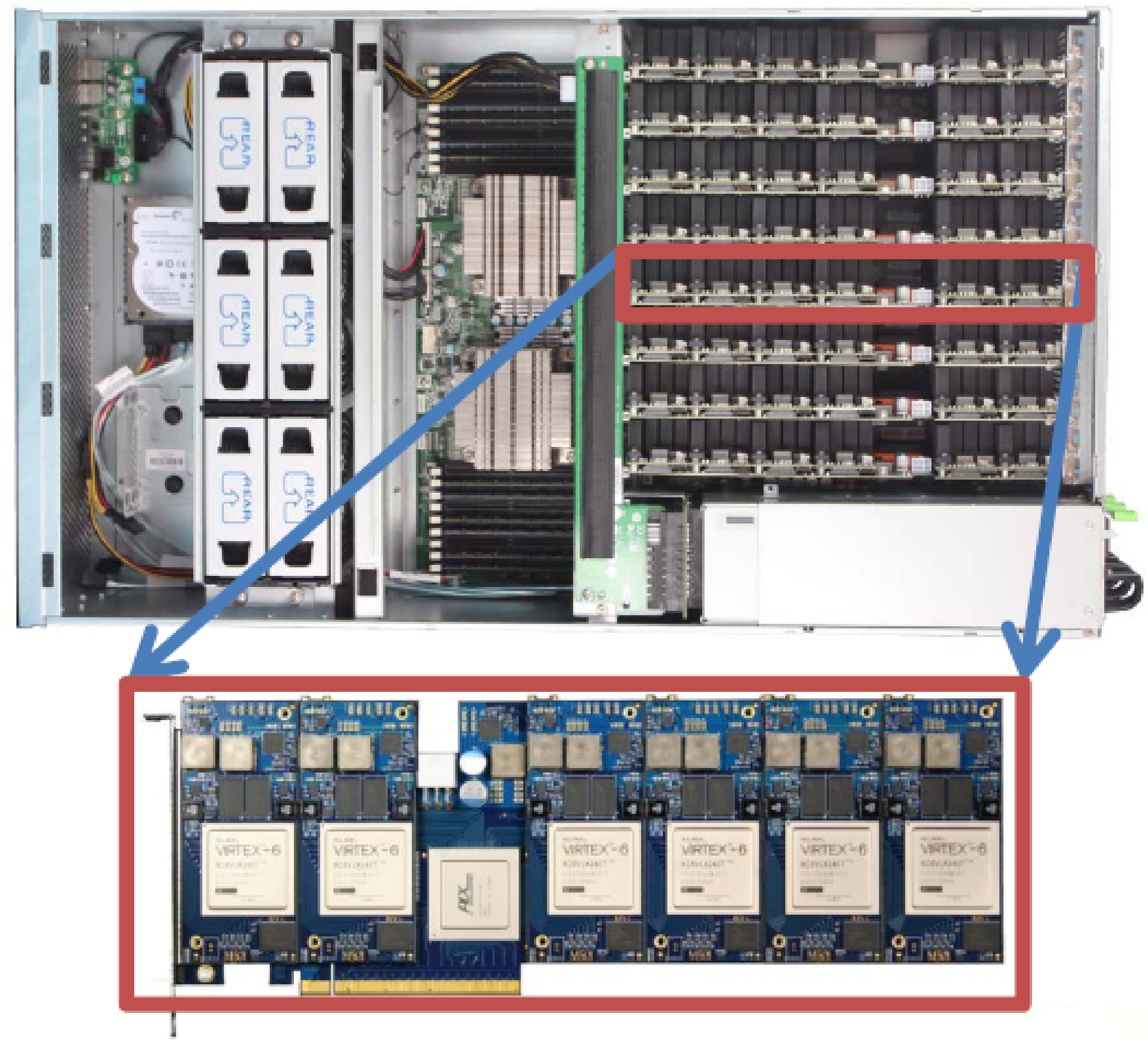
- Many Integrated Core (MIC) architecture
  - Based on the never-released Larrabee
- Over 1 TFLOPS performance
- Already in use in #150 on June 2012 TOP500 list
  - Soon also in a would-be-#3, but not generally available yet
- 50 to 64? x86 cores based on the original Pentium
- Per core: 512-bit SIMD unit, 32 KB L1 data cache, ...
- 8 GB GDDR5 RAM on a 512-bit bus?
- Can run an almost standard OS, yet is a coprocessor
- Programmed like a multi-core CPU rather than a GPU

Xeon Phi for password hashing

- Likely more straightforward than defensive use of GPU
- Should be able to efficiently run 8 bcrypt instances in SIMD vector elements per core (as limited by 32 KB L1 cache, leaving half the SIMD vector width unused), thus 400+ concurrent instances per chip at ~2.0 GHz
  - May be the bcrypt killer, especially if it is affordable
- For defensive use, should run a hash function with sufficient parallelism at least to use a 512-bit vector
- Probably can't hold a host-unreadable local parameter
  - Not intended as a security device, almost certainly allows DMA

Pico Computing's FPGA cluster

- 6 Virtex-6 or Kintex-7 FPGAs per board
- 8 PCIe boards per 4U chassis (48 FPGAs)
- Up to 192 GB DDR3 RAM on FPGA boards
- 3x1200W PSUs (N+1)
- Dual quad-core Xeon
- Up to 144 GB RAM



SC5 SuperCluster with 48 M-501 modules

Image (c) Pico Computing, reproduced under the fair use doctrine

Password hashing on FPGAs

- Password cracking on FPGAs had been done before
  - Including by Pico Computing
- We explored possible defensive use of FPGAs, as well as bcrypt cracking on FPGA
  - Yuri Gonzaga's Google Summer of Code 2011 project

    Co-mentors: Solar Designer (Openwall), David Hulton (Pico Computing)
  - Yuri wrote and debugged Verilog code implementing bcrypt (including with Block RAMs), multiple bcrypt cores per chip, a tiny bcrypt-like construct (with intent to explore the possibility of fitting hundreds or thousands of these per chip)
  - We also considered reuse of Pico's fully-pipelined DES cores

bcrypt on FPGA

- Blowfish S-boxes fit Xilinx Block RAMs perfectly
  - Can't reasonably use pipelining, but can improve resource usage by implementing multiple instances of bcrypt per core
    - (not completed in the GSoC project)
- Low clock rate, thus high latency (compared to CPU)
- Reasonable throughput may be achieved due to large number of cores
- Estimate: optimal implementation on Pico's M-501 (one Virtex-6 LX240T) could be ~5x faster than optimal code on quad-core CPU (without AVX2)

FPGAs for defense

- To have maximum advantage over CPU and GPU, number of cores times number of pipeline stages (if applicable) should be maximized
  - Thus, each core should be as small as possible

    Rationale: CPU has a limited number of relatively feature-rich execution units. By having very simple cores, we leave more logic in the execution units unused. Difficulty: SIMD instructions may operate on many narrow bit width values in parallel. A way to defeat implementation of small S-boxes with SIMD byte permute instructions (in Cell, SSSE3, XOP) or with bitslicing is through making the S-boxes variable, but parallel S-box lookups may nevertheless be performed with gather loads (in AVX2 VSIB, to be available in 2013+).
- Alternatively, focus on making optimal use of resources without trying to slow down CPU/GPU implementations

Local parameter in FPGA

- To avoid specifying it at synthesis, may be patched into a Block RAM's initial state in bitstream
- Bitstream may be stored in a flash memory chip
  - Loaded into FPGA from flash on power-on
  - Only a subset of Pico's boards have flash
    - Others have to be configured from host before use
- Not host-unreadable in existing boards as-is
  - May be retrievable via partial reconfiguration
- A hardware revision may be made
  - e.g., a jumper to enable configuration mode
  - Pico would do it if there's demand

Takeaways

- Salting and stretching are a must, but you knew that
- Unreadable local parameter is also a must for large user/password databases - need two extra devices
- HSMs might (not) be safer than regular machines
- PBKDF2 is not good enough unless we're on GPU
- Use of hardware beyond CPU + RAM for password stretching is tricky and currently not obviously beneficial overall (considering extra R&D, risks, cost) - further research and experiments are needed

Consult a doctor

Design and implementation of a password hashing setup is serious business

- Larger organizations (with millions of users or with particularly high-value accounts - e.g., online banking) may benefit from custom setups, but independent review by a qualified consultant is a must
- Smaller organizations are better off using pre-existing solutions
  - Currently this means straightforward use of bcrypt

    A short-term recommendation only, unfortunately
  - Independent review is highly desirable, but is not crucial

Questions?

Solar Designer <solar@openwall.com>

@solardiz @Openwall

http://www.openwall.com