Understanding Optimizations and Measuring Performances of PBKDF2

Andrea Francesco Iuorio

11 February 2018

Introduction

Introduction to KDFs

Human users needs to interact with several cryptographic systems on a daily basis

Problem: these systems uses cryptographic keys as authentication secrets!

Human users however prefer to use passwords

- Easier to provide for authentication
- Easier to remember

Introduction to KDFs (cont.)

Why don't we use passwords as cryptographic keys?

Because passwords are terrible keys!

- Passwords are often short and have variable length, while keys are long and fixed-length
- Passwords usually have too low entropy, while keys must possess high entropy

Key Derivation Function (KDF): an algorithm that converts passwords into secure keys

PBKDF2

$$Key = T_1||T_2||\ldots||T_{len},$$

$$T_i = U_1 \oplus U_2 \oplus \ldots \oplus U_k,$$

$$U_1 = PRF(p, s||i),$$

$$U_c = PRF(p, U_{c-1})$$

The main ideas behind the design of PBKDF2 are:

- The use of a pseudorandom function (PRF) and salt
- The iteration of ”k” rounds of the PRF

PBKDF2 (cont.)

RFC 8018 (January 2017) suggest these defaults:

$$PRF = HMAC - SHA1$$

$$k = 1000$$

- HMAC-SHA1 is still considered secure as a PRF

These values are enough for general purpose applications

Our goals

Our main question:

- For which kind of applications can we consider PBKDF2-HMAC-SHA1 secure today?

We split this question into:

- What kind of impact can optimizations to the internal PRF bring to PBKDF2?
- What are the computational possibilities offered by current consumer-grade hardware?

Our procedure

To answer these questions, we:

- studied PBKDF2, HMAC and SHA-1 and the state-of-the-art of their implementations
- developed from scratch two implementations for PBKDF2: one CPU based and one GPU based
- performed a testing activity, comparing results with other implementations

Optimizations rundown

SHA-1 word expansion

SHA-1 works in blocks of 512 bits (16 words)

The block input is expanded into 80 words using this equation

$$W_i = ROTL^1(W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16})$$

However, this equation can be replaced by

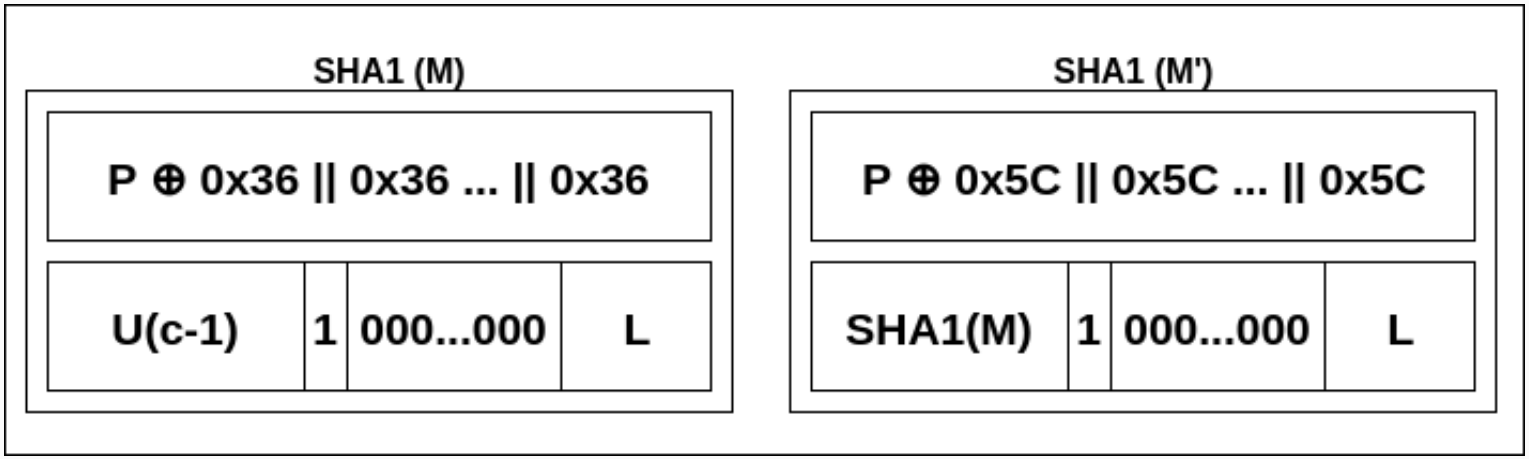
$$W_s = ROTL^1(W_s \oplus W_{(s+2) \wedge MASK} \oplus W_{(s+8) \wedge MASK} \oplus W_{(s+13) \wedge MASK})$$

$$s = i \wedge MASK; MASK = 0xF$$

We only need 16 words instead of 80 for the input expansion!

HMAC block reduction

$$HMAC(P, U_{c-1}) = H(P \oplus opad || H(P \oplus ipad || U_{c-1}))$$



But the first 2 blocks are always the same for each U_i!

- We can precompute those SHA-1 blocks
- Reduces the number of SHA-1 blocks from $4 * c$ to $2 + 2 * c$

Our GPU implementation

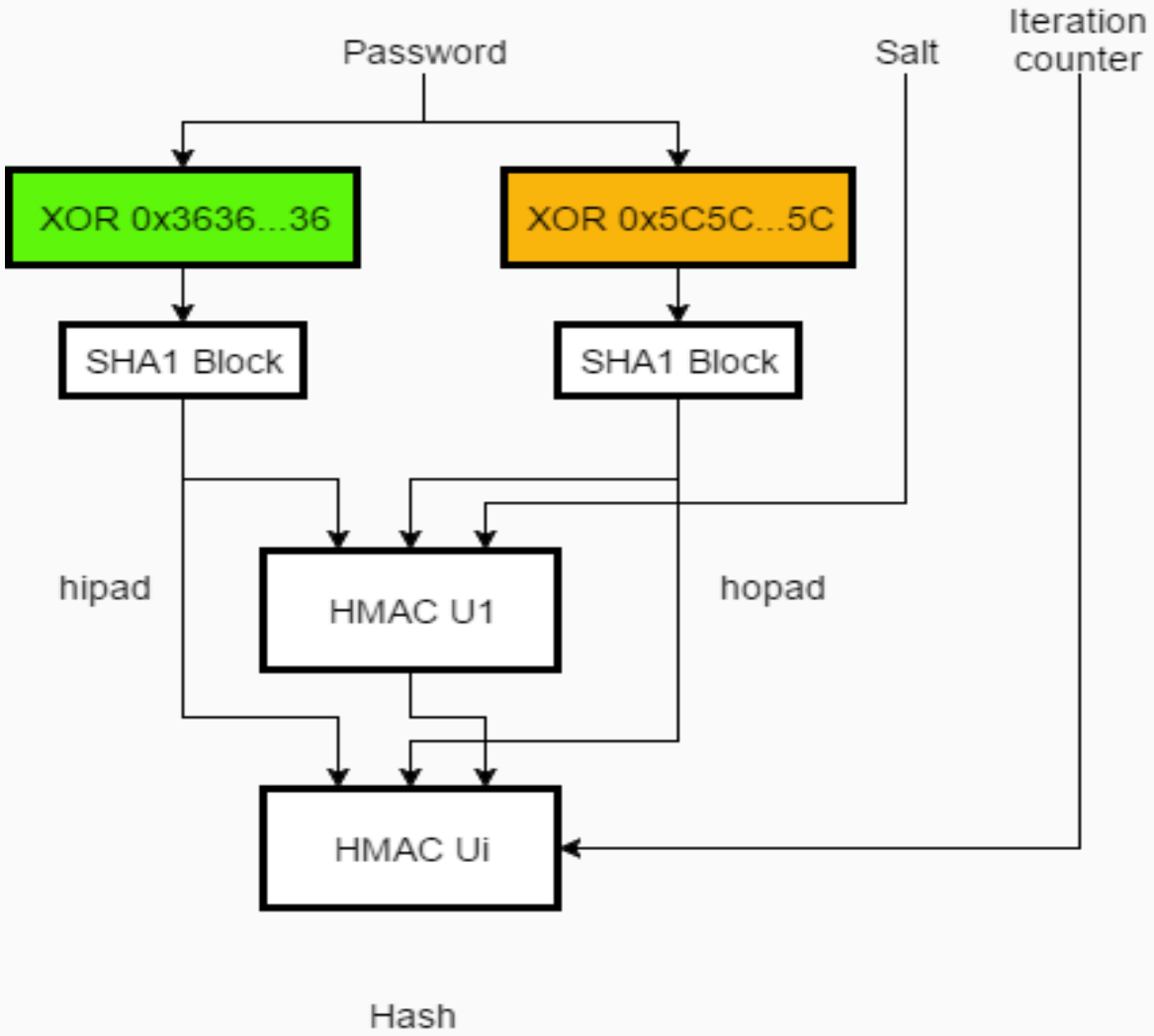
GPU implementation

We developed from scratch an highly optimized implementation of PBKDF2 using OpenCL

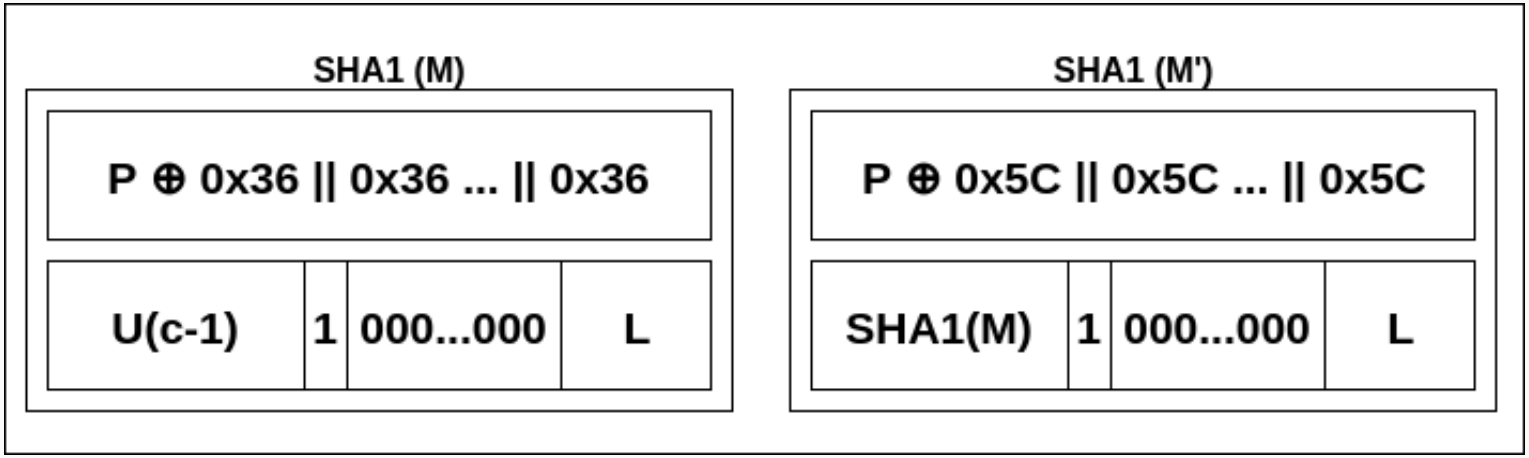
We have implemented PBKDF2 with two OpenCL kernels:

- A kernel that executes only the first iteration of HMAC
- A kernel that executes the remaining $i - 1$ iterations

GPU implementation (cont.)



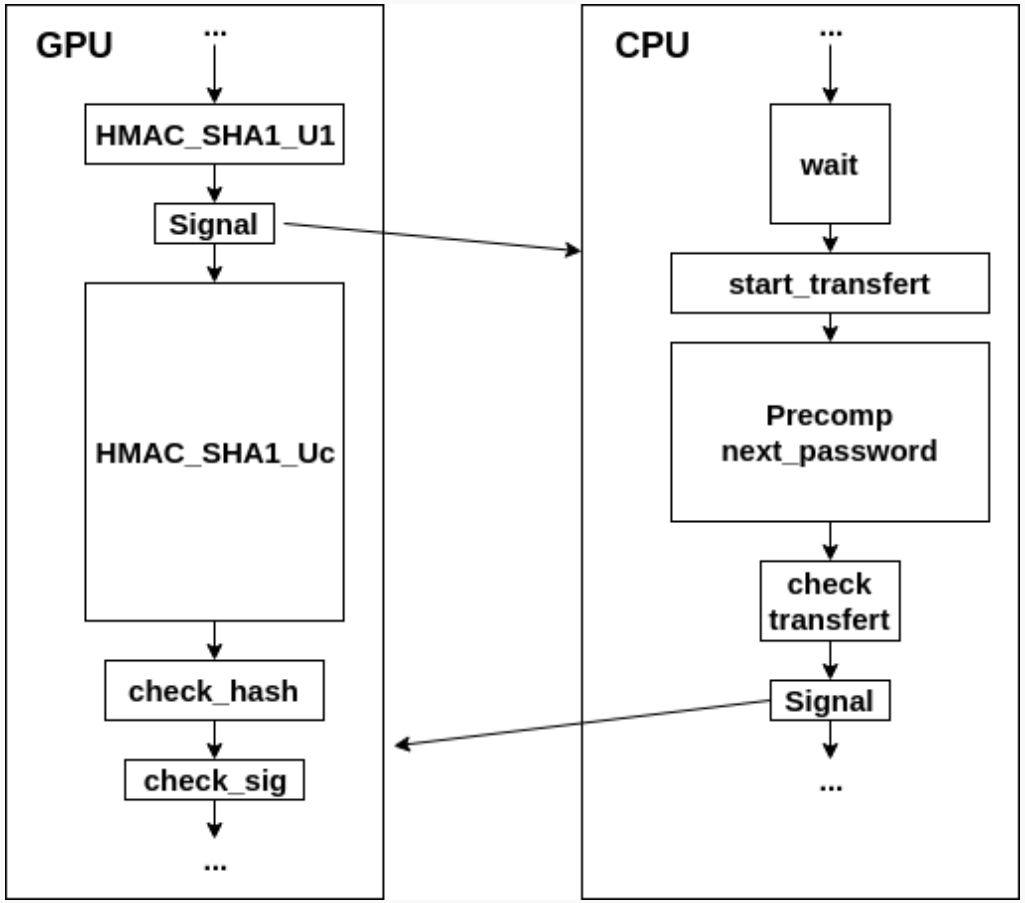
GPU implementation (cont.)



Because of the HMAC optimization we've seen earlier, passwords are only needed during the first HMAC iteration!

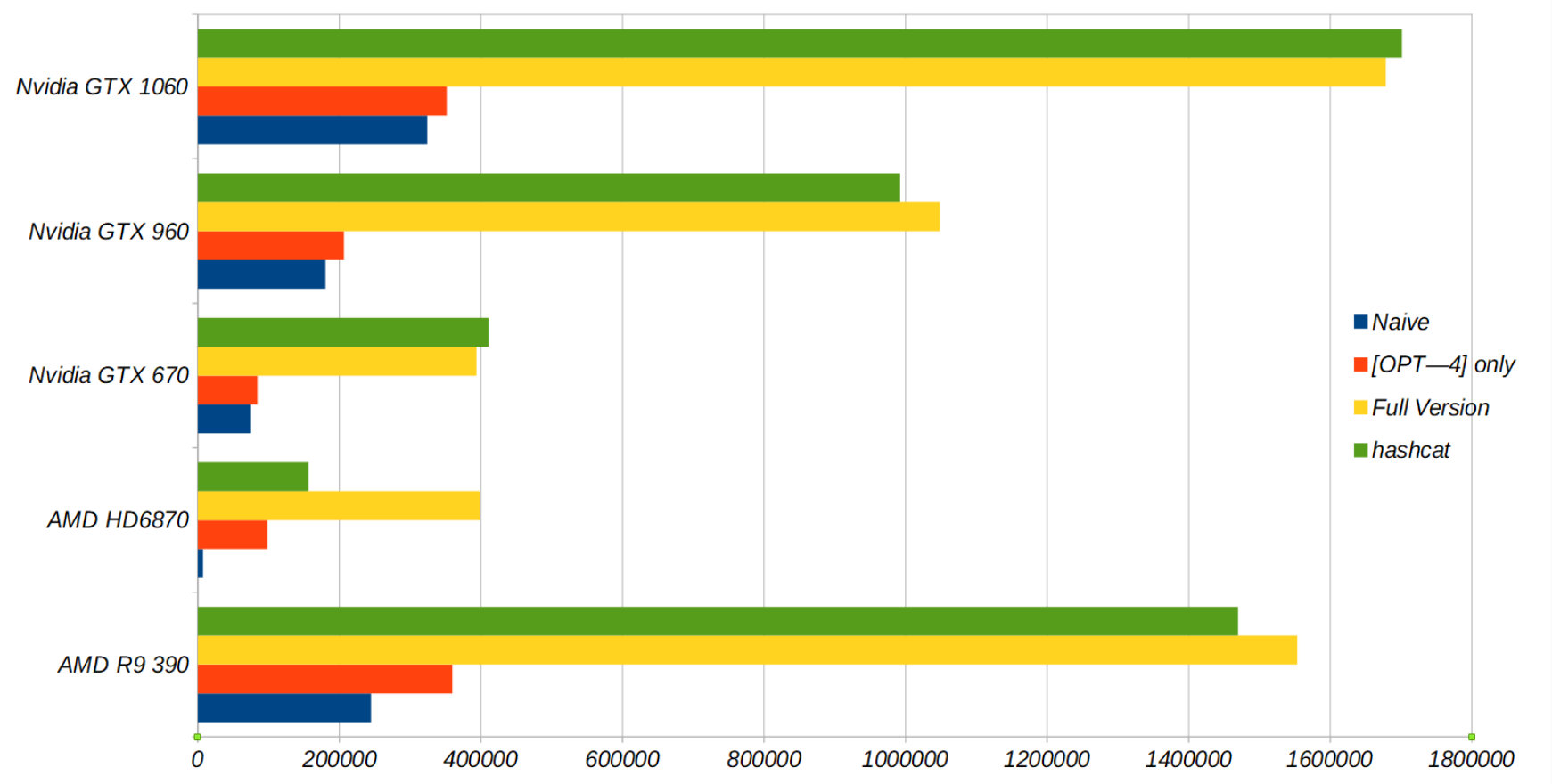
- We can update the GPU memory containing the passwords while it executes the second kernel without race conditions

GPU implementation (cont.)



Testing activities

Testing results



What we have learned

- Some optimizations only works on CPUs, others only on GPUs
  - Example: SHA-1 word reduction worked only on GPUs
- Consumer-grade GPUs are able to compute enough passwords to attack a human-generated secret
  - 15 minutes: a dictionary-based password
  - 5 days: a 6 character long password
  - 11 days: the average human-generated password (8 character long, 40.54 bits of entropy)

Conclusions and future works

Conclusions

To generate secure enough keys, PBKDF2-HMAC-SHA-1 needs

- Longer and better input passwords
- An high iteration count

Both however present usability problems for human users!

The use of CPU-intensive operations as a brute-force resistance technique is undermined by cheap parallel architectures like GPUs and ASICs

Future works

In 2015 the Password Hashing Competition selected new KDFs like argon2

Their brute force resistance is based on new techniques like memory-hard functions

$$S(n) \in \Omega(T(n)^{(1-\epsilon)})$$

Future works could include the analyses of the strength of these crypto functions.