Yandex

Yet another password hashing talk

Evgeny Sidorov

Make out my accent

- Yescrypt
- Scrypt
- Bcrypt

Agenda

1 INTRO

2 Algorithms

3 Protocol designs

4 Conclusions

Introduction

Intro

- Surprisingly passwords are still alive
- Passwords will probably always exist
- “Something that you know” in 2FA

Password Hashing

› Non-interactive operations (password based encryption etc.)

› Interactive / online operations (password based authentication in a web service etc.)

› This talk is about interactive / online operations

Attacker and Defender

- Defender uses traditional multicore server CPUs (bigger cache per core, bigger RAM available)
- Attacker can use GPUs, FPGAs, ASICs
- GPU is the most important case as they're ready to use

Algorithms

The role of PHA (interactive mode)

- Itself not the perfect security measure
- Must be considered as a “last hope”
- Must give users time to change their passwords in case of DB compromise (total compromise)

Strategies behind PHAs

- Optimise as much as possible for CPUs, their capabilities and RAM access patterns
- Make GPUs, ASICs, FPGAs ineffective/expensive by using more RAM (and other of their bottlenecks)
- Memory hard - trading memory for computations leads to much bigger time and computational costs

Yescrypt

Yescrypt

- Designed by Solar Designer (aka Alexander Peslyak)
- Got special recognition from PHC
- Optimised for “password hashing at scale”

Yescrypt

- Fixes scrypt's TMTO
- Uses not only random reads but also random writes (increases the amount of recomputation)
- Replaces Salsa20/8 with custom "pwxform" to achieve bcrypt-like resistance to GPUs
- Some tweaks in parallelism (moved to deeper level)

Scrypt


PBKDF2-HMAC-SHA-256
SMix
BlockMix
Salsa20/8
PBKDF2-HMAC-SHA-256

pwxform

- An optimisation for low memory case
- Uses 12kb s-boxes (8kb active lookups, 16 byte individual lookup) and it’s friendly for CPU L1 cache
- Uses integer multiplications (higher latency on FPGAs/ASICs)
- Friendly to SIMD instructions

Pros&Cons

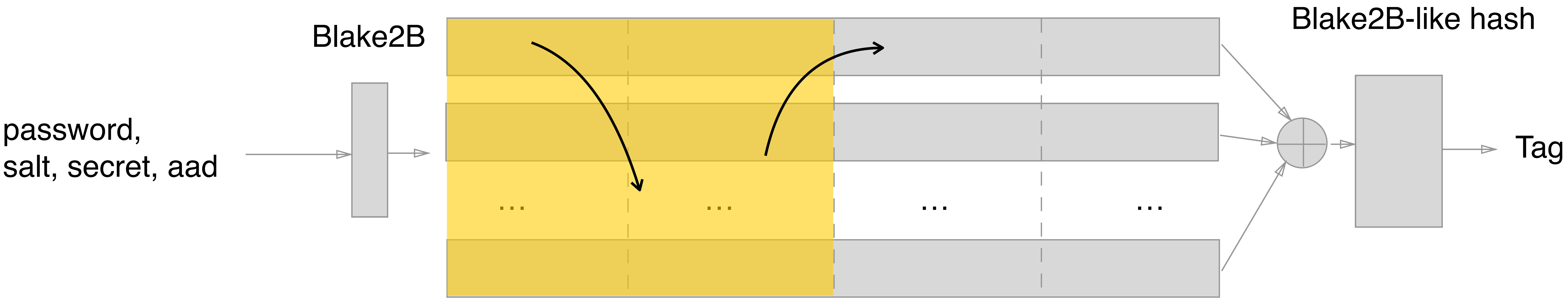
- Optimised for low memory case (bcrypt-like resistance)
- Fixes Scrypt drawbacks
- Pretty complex
- There is no resistance to side channel attacks

Argon2

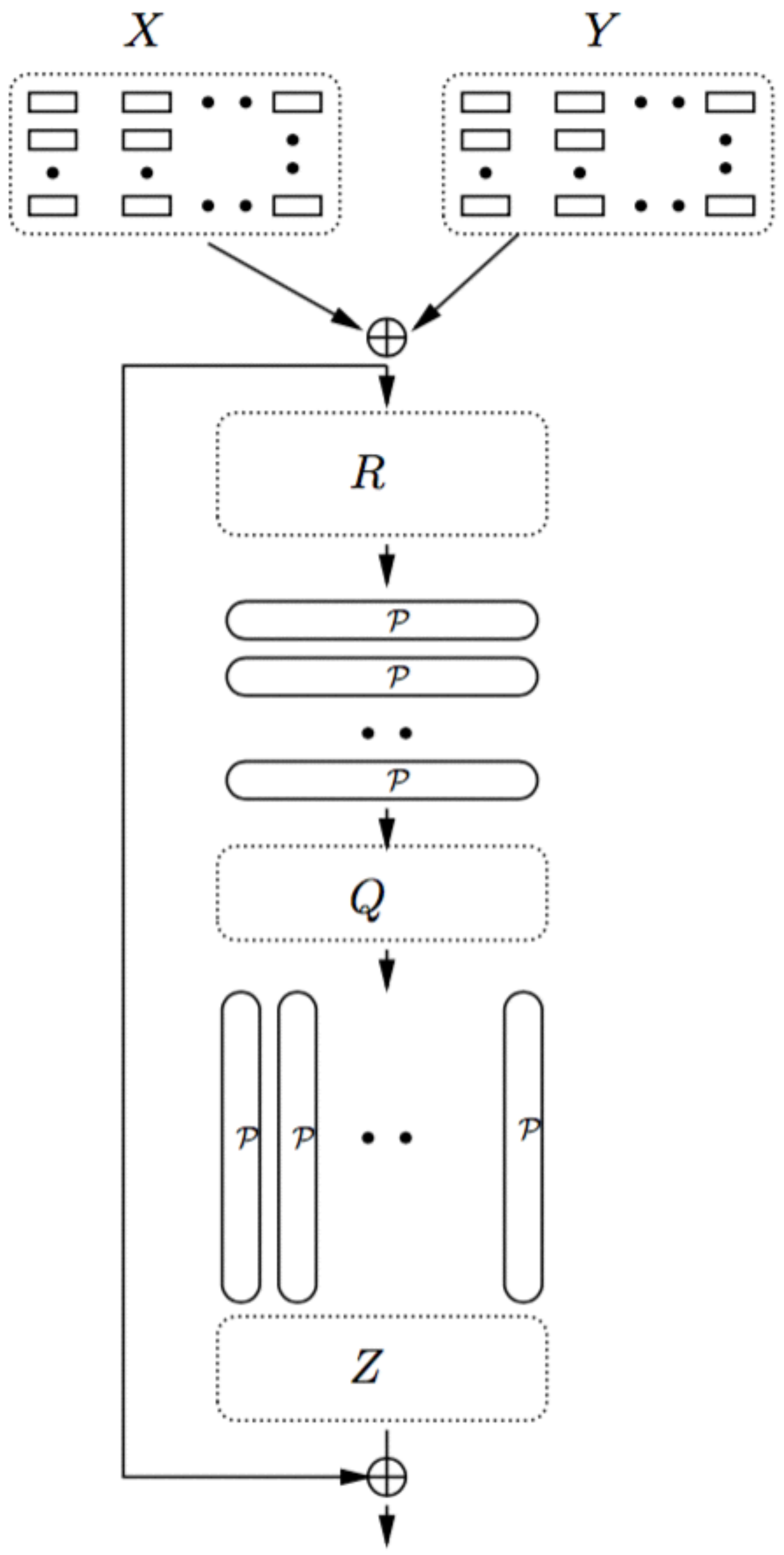
Argon2

- Simple, tuneable & secure
- Three versions: i, d, id

Argon2



Argon2id, first pass



Compression function

$$
\begin{aligned}
a &\leftarrow a + b\\
d &\leftarrow (d \oplus a) \ggg 32\\
c &\leftarrow c + d\\
b &\leftarrow (b \oplus c) \ggg 24\\
a &\leftarrow a + b\\
d &\leftarrow (d \oplus a) \ggg 16\\
c &\leftarrow c + d\\
b &\leftarrow (b \oplus c) \ggg 63
\end{aligned}
$$

(a) Blake2 G function.

$$
\begin{aligned}
a &\leftarrow a + b + 2 \cdot \mathrm{lsw}(a) \cdot \mathrm{lsw}(b)\\
d &\leftarrow (d \oplus a) \ggg 32\\
c &\leftarrow c + d + 2 \cdot \mathrm{lsw}(c) \cdot \mathrm{lsw}(d)\\
b &\leftarrow (b \oplus c) \ggg 24\\
a &\leftarrow a + b + 2 \cdot \mathrm{lsw}(a) \cdot \mathrm{lsw}(b)\\
d &\leftarrow (d \oplus a) \ggg 16\\
c &\leftarrow c + d + 2 \cdot \mathrm{lsw}(c) \cdot \mathrm{lsw}(d)\\
b &\leftarrow (b \oplus c) \ggg 63
\end{aligned}
$$

(b) BlaMka G function.

BlaMka function

https://eprint.iacr.org/2015/136.pdf

Argon2 design principles

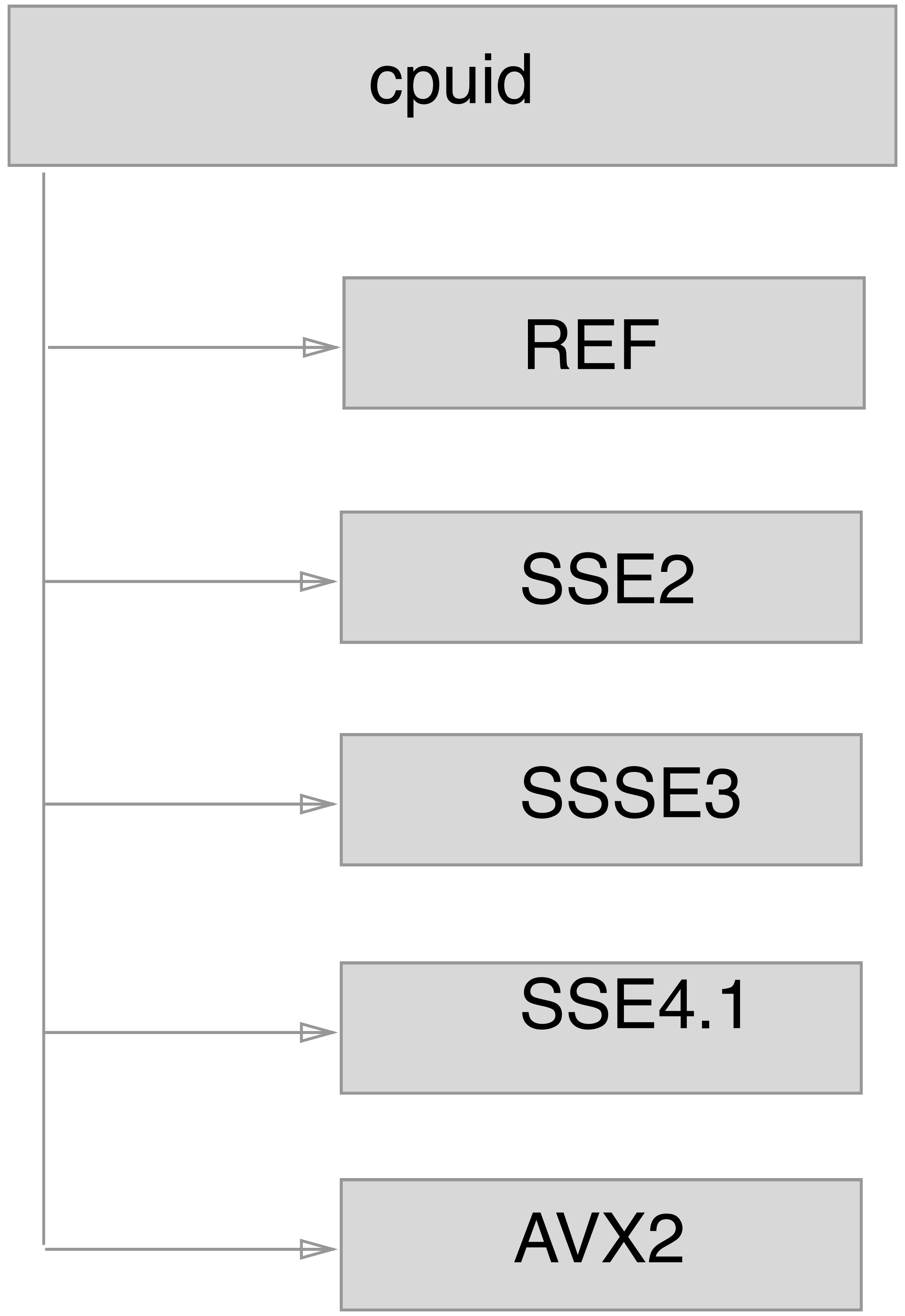
- Operates on 1kb memory blocks
- Optimised for x64, SIMD friendly
- Uses Blake2B and BlaMka as underlying primitives
- BlaMka increases the circuit depth and makes ASIC implementation more complex (uses integer multiplication)

PHC implementation

- PHC implementation of Argon2 uses “-march=native”
- Can easily run up against “Illegal instructions exception”
- Compatibility costs 15-20% of performance

Our implementation

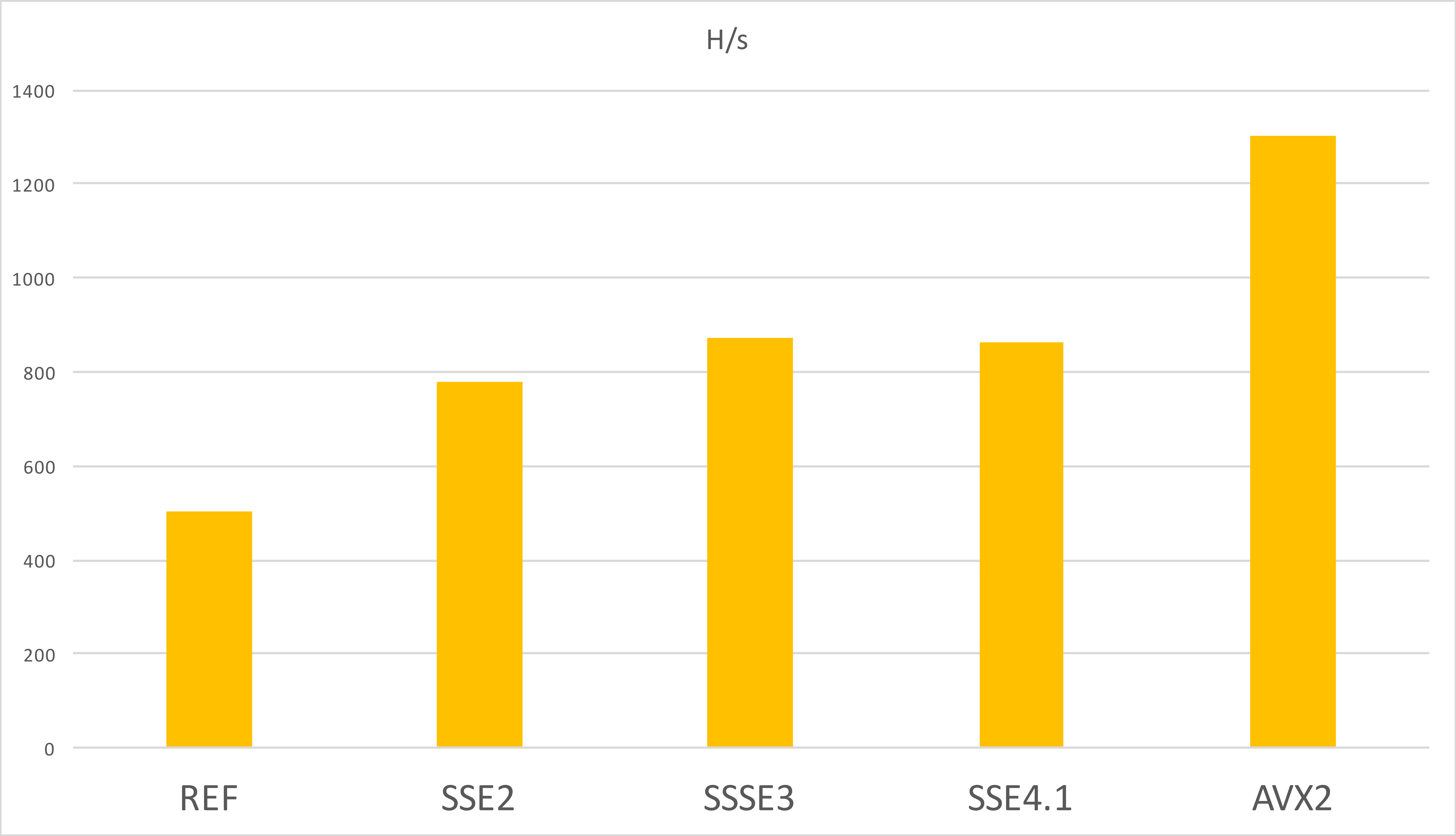
- Employs SIMD as much as possible
- Runtime CPU dispatching to run the version optimised for current CPU
- Makes deployment easier (especially for clouds)
- SIMD-optimized Blake2B



Naming

Argonishche = Argon + itch (-ищ)

Benchmark Argon2d (1, 2048, 1) c4.8xlarge



Argonishche (Argon2 implementation)

- https://github.com/yandex/argon2
- C++14 (constexpr, partial templates etc.)
- BSD 3-clause license
- AVX512 (Intel Skylake) is in plans

Protocol Designs

Motivation

- Algorithms are not enough
- Large parameters are hardly affordable for high-loaded services
- Offline attacks are possible
- Need to apply protocol level mitigations

Protocol designs

- Local parameters
- Big ROMs (“security through obesity”)
- Crypto Anchors
- Partially Obvious PRFs

Local parameters

- Add secret key to password hashing algorithm
- Secret key stored in the application conf, not in the DB
- SQL injection or just access to production DB become insufficient
- RCE in the application is needed

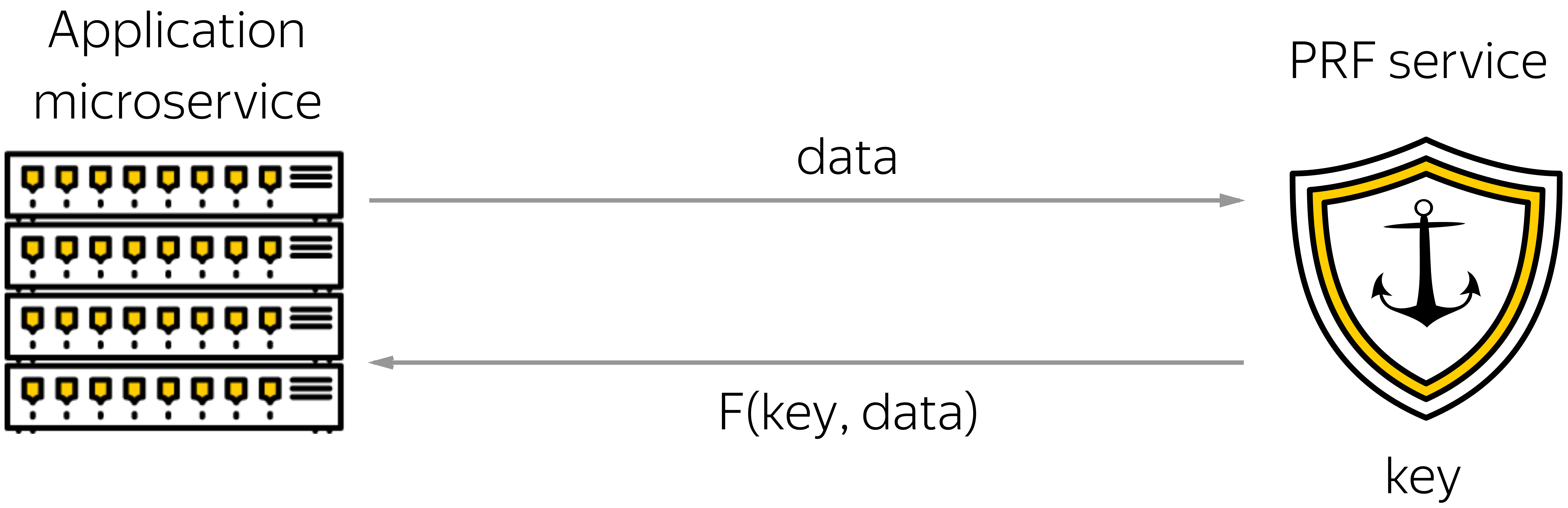
Big ROM pros

- A relatively big pre-filled shared memory area
- using many cores more expensive for the shared ROM, less cores are good (like in a CPU)
- For example, 112Gb ROM is harder to steal

Big ROM cons

- Make it harder to switch to a new PHA (you’ll have to live with the ROM)
- ROM-as-as-service amortisation approach (pieces of ROM can be scattered around less powerful servers, network is used to query values)
- Leads to additional steps in case of server reboot etc.

Crypto Anchors



Crypto Anchors

- Tiny PRF services (can apply HMAC for example)
- Turn offline attacks into online attacks (attacker just can’t go home and have fun with hashes)
- In case of HMAC key updates are impossible (need to add one more layer)

PASS: PRF service

PASS: Strengthening and Democratizing Enterprise Password Hardening

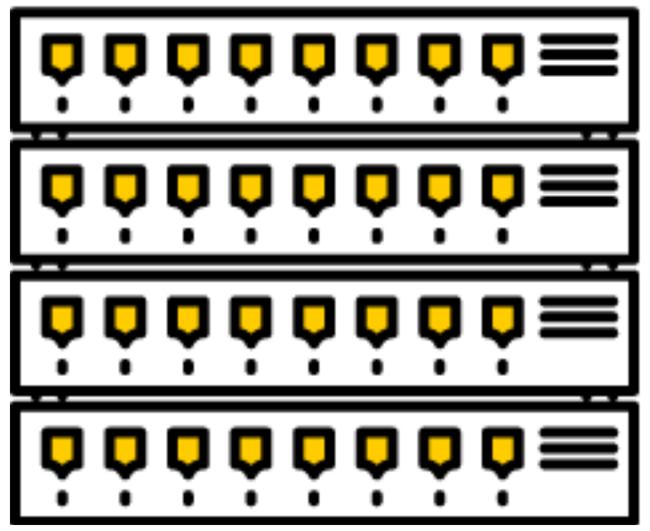
Ari Juels
Jacobs Technion-Cornell Institute
Cornell Tech

with D. Akhawe (Dropbox). A. Athalye (MIT), R. Chatterjee (Cornell), A. Everspaugh (UWisc), T. Ristenpart (Cornell Tech), S. Scott (Royal Holloway)

$$e: G_1 \times G_2 \to G_T$$
$$e(a^u, b^v) = e(a, b)^{uv}$$

Bilinear pairing

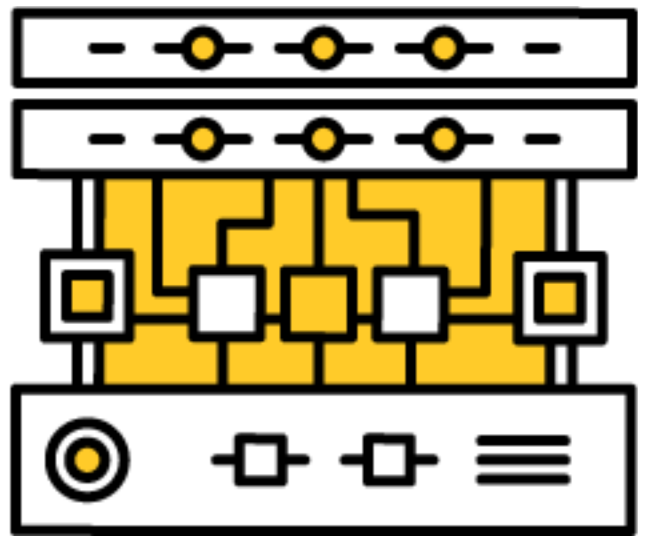
Blinding $x := H(P)^r$



t, x

y

$y := F_k(t, x) = e(H(t), x)^k$



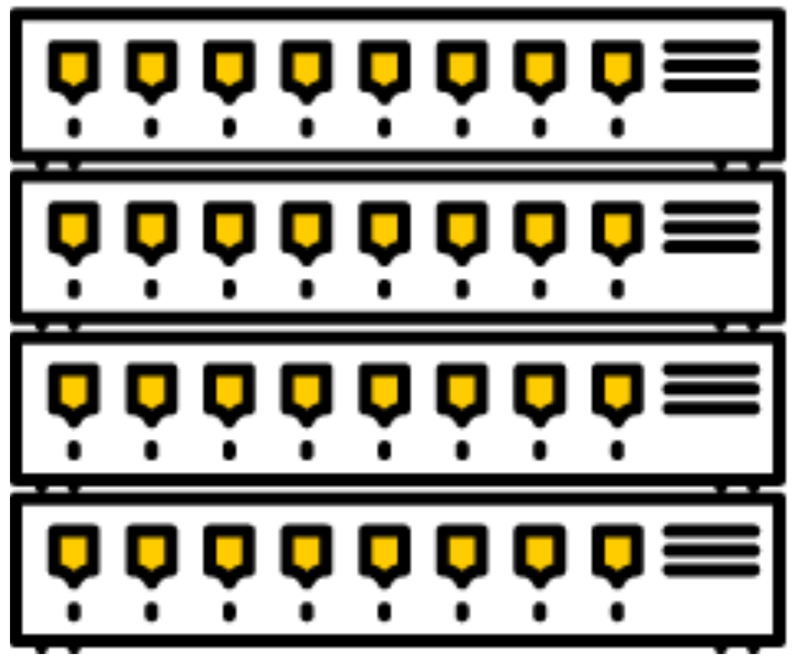
PO-PRF service

$$z := y^{\frac{1}{r}} = e(H(t), H(p))^{k \cdot r \cdot \frac{1}{r}}$$
$$= e(H(t), H(p))^k$$

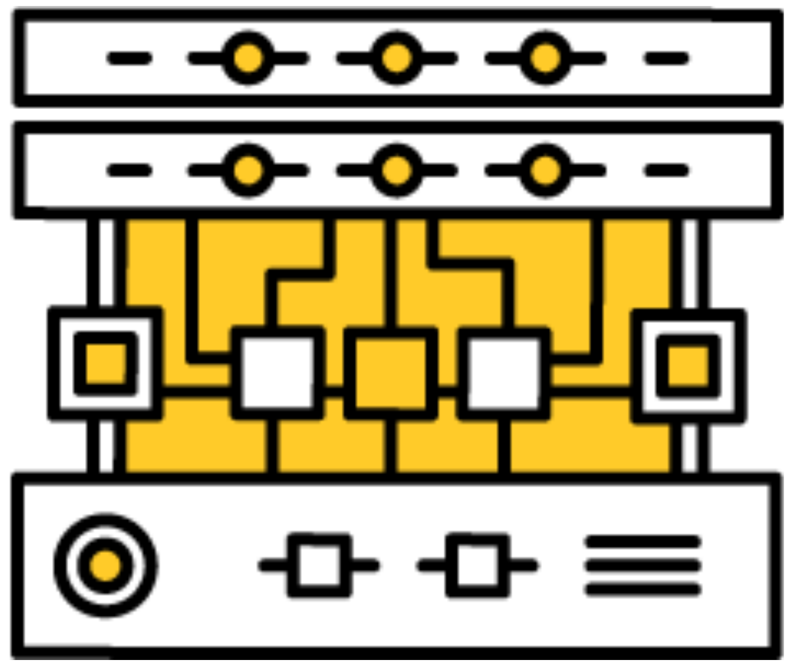
Unblinding

Key rotation

PO-PRF service



$$\Delta_{k \to k'} = k'/k$$



$$z' := z^{\frac{k'}{k}} = e(H(t), H(p))^{k * \frac{k'}{k}} = e(H(t), H(p))^{k'}$$

Pythia PRF

- The idea is like those in PASS
- Uses Zero Knowledge Proof scheme to proof correctness of calculations
- Multitenancy

Conclusions

Conclusions

- Good algorithms are not enough for online password hashing
- There are some good approaches to make up for it
- Offline attacks can be turned into online attacks

Acknowledgements

- Dmitry Khovratovich for useful discussions about Yescrypt and Argon2
- Alexander Peslyak for discussions about Yescrypt
- Igor Klevanets for reviewing the Argon2 library (Argonische)

Useful links

- https://eprint.iacr.org/2015/644 Pythia PRF Service
- https://github.com/yandex/argon2 Yandex’s Argon2 impl
- http://bit.ly/2vS0I7B Pythia ref. implementation

Questions?

Evgeny Sidorov

Information Security Engineer

e-sidorov@yandex-team.ru

https://github.com/yandex/argon2