Is Password InSecurity Inevitable?

Cryptographic Enhancements to Password Protocols

See correction in slide 19

Hugo Krawczyk (IBM Research)

Works with Stanislaw Jarecki, Jiayu Xu (UC Irvine)
Aggelos Kiayas (U Edinburgh)
Nitesh Saxena, Maliheh Shirvanian (UA Birmingham)

Real World Crypto 2017

Password (In)Security

- Passwords: **MAIN** authentication tool in the digital era
- Protect our lives and social order, *conveniently* and **Insecurely**
- BILLIONS of passwords stolen
  - Yahoo 500M, MySpace 360M, LinkedIn 165M, eBay 145M,…, Ash-Mad 11M
  - … Twitter, RSA, Google, Dropbox, PayPal, Sony, …
- https://www.leakedsource.com:
  - 2,918,283,623 accounts at your service.
  - "Check for free to see if your email or account was hacked."

An Unacceptable State of Affairs (but do we have a choice?)

- Unacceptable, really! Our social order depends on passwords
- But do we have a choice?
  - Get rid of passwords all together: Not realistic, too convenient, massively deployed.
  - Ask users to memorize (multiple) high-entropy passwords: No way
  - Stop choosing same/related password: No way

Can Cryptography Help?

- **Yes**!
- We show strong password protocols for a variety of problems in a variety of settings.
- Using simple, well-established techniques
  - Mostly blinded DH [Chaum, Ford-Kaliski, Boyen, …] ("oblivious PRF")
- Efficient. Mature. **Ready for deployment** in the **real world.**
- I will go over three such solutions very briefly.
- Pointers to papers at the end; and please talk to me if you are interested to learn more (esp. if you can transfer this to practice).

Offline Dictionary Attacks

- Main source of password compromise:
  - *Deadly combination* of human memory limitation (→low entropy passwds) and server compromise
  - Attacker that gets hold of "password file" can test candidate passwords against stored hashes; cost proportional to dictionary size
- The most effective attack on passwords
  - Millions++ of passwords tested per second (from s/w to dedicated h/w)

☹ Offline attacks upon server compromise are **unavoidable**

  - If the server can check, so does the attacker

Hope: Make these unavoidable exhaustive attacks ineffective

(High-entropy passwords or additional devices/servers)

Part I:

Take the burden of choosing and memorizing passwords off humans

A simple solution: Password Store (a.k.a. password manager)

- Carry strong independent passwords stored
- … in your phone, your smart watch, …, or retrievable online
- … encrypted under *a master password*
  - Just remember the one master password (hopefully non-trivial)

Graphic zoho.com

Password store: Not without problems

- A list of user passwords encrypted under the user's master password
  - Attacker obtains the list ➔ offline attack on master password (all the user's passwords compromised)
  - "Inside compromise": Attacker learns master password as user types it
  - User-device communication compromise (master password leaked)
  - Furthermore: Typical password managers keep *user-chosen passwords* (hence, weak and related/repeated)
- Can we do better?

A dream password store

- All passwords kept in a password store in a *user's device or online*
- User memorizes a *single* master password
- All passwords are *random* and *independent* of each other
- And: An attacker getting hold of store or even in full control of the device… **learns nothing**
  - About the individual stored passwords
  - Or the master password

What do you mean by *nothing* ?

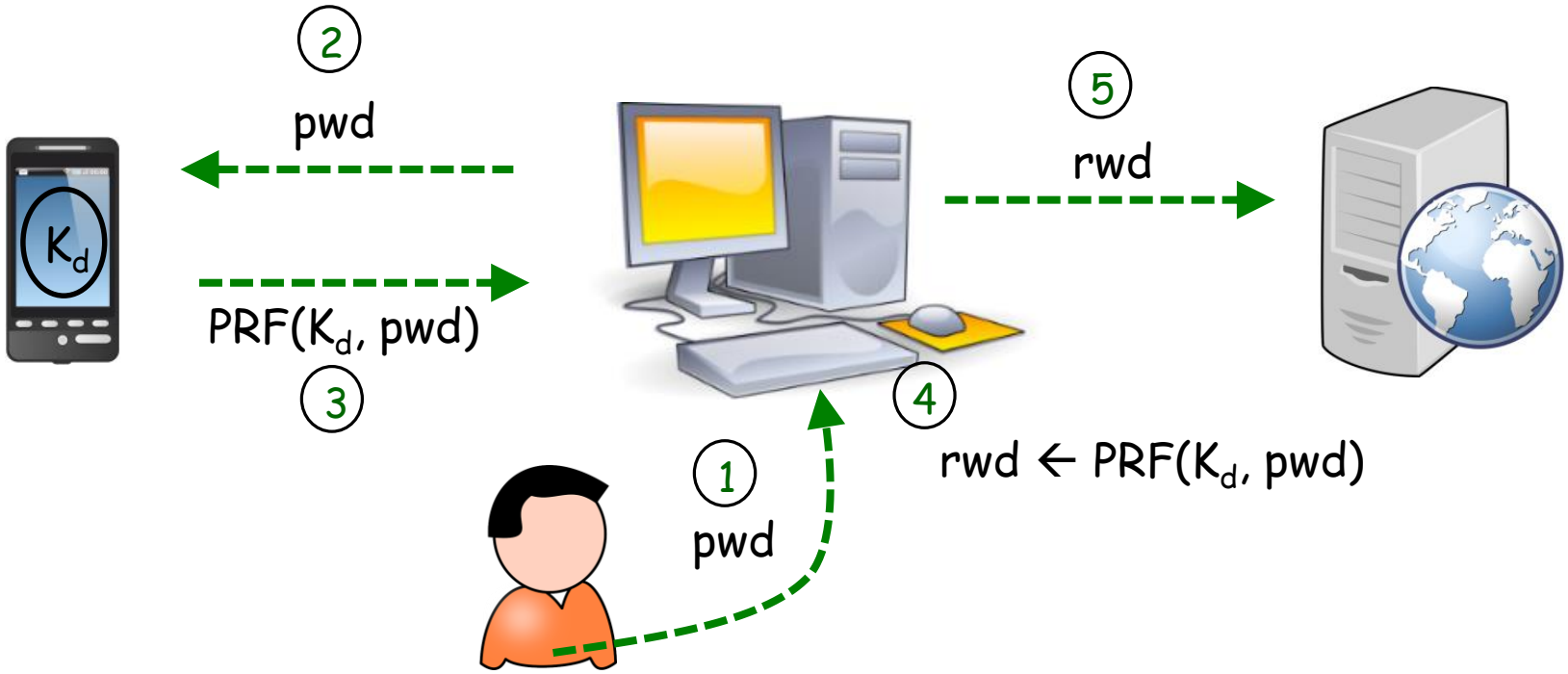
- Well… *nothing*. As in information-theoretic nothing!
  1. Information stored in the device is *independent of the user's individual passwords and independent of the master password*
  2. Cleartext master password is *never entered* into the device!
     - An eavesdropper or active attacker on the link to the device learns nothing
     - An attacker inside the device, w/full control, even when user enters the master password does not learn anything either (not even at init!)

SPHINX: A password **S**tore that **P**erfectly **H**ides from **I**tself (**N**o **X**aggeration)

SPHINX

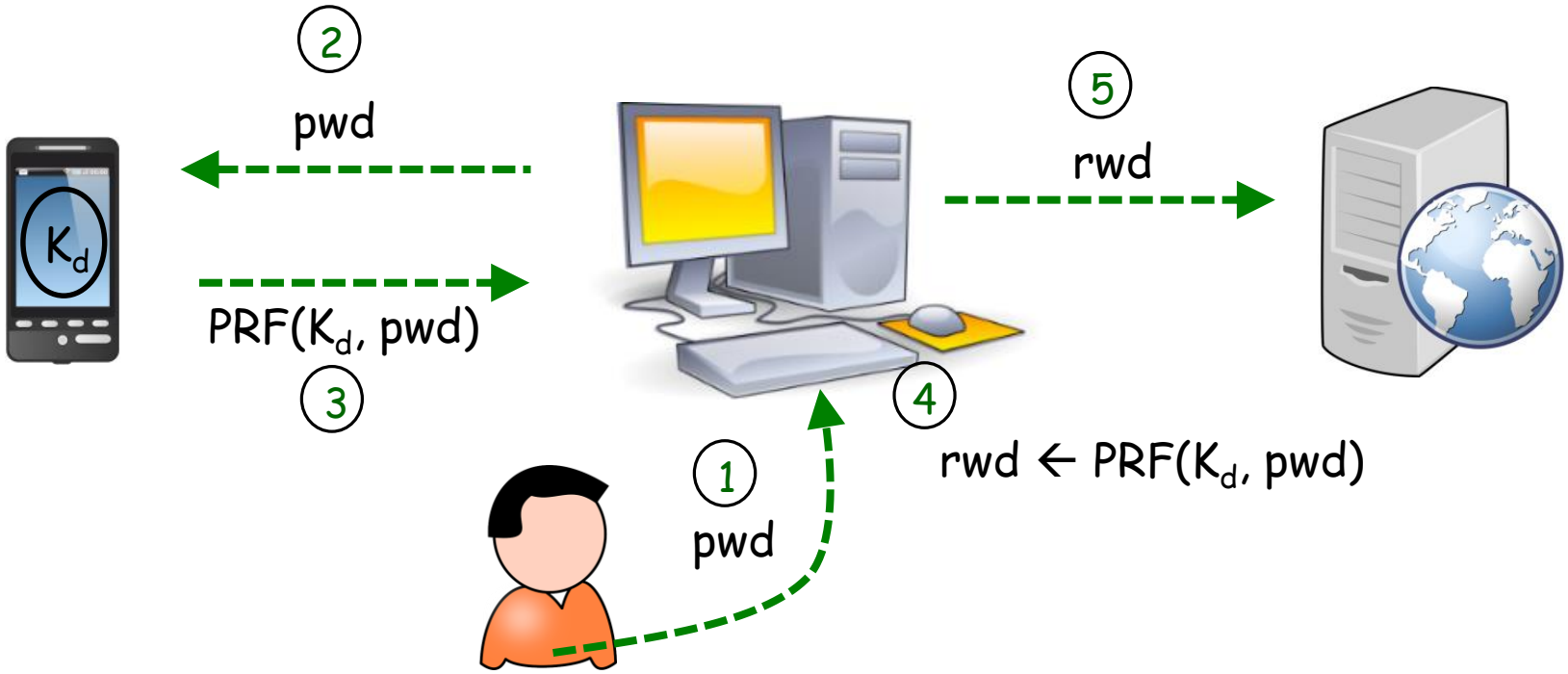
- A password Store that Perfectly Hides from Itself
- Really? Let me show you.

PRF-based Solution



- rwd is a (pseudo) random password that user registers with server
- There is an independent rwd with each service, e.g. $PRF(K_d, pwd \mid url)$
- **Works with *any* password protocol between client and server**

PRF-based Solution

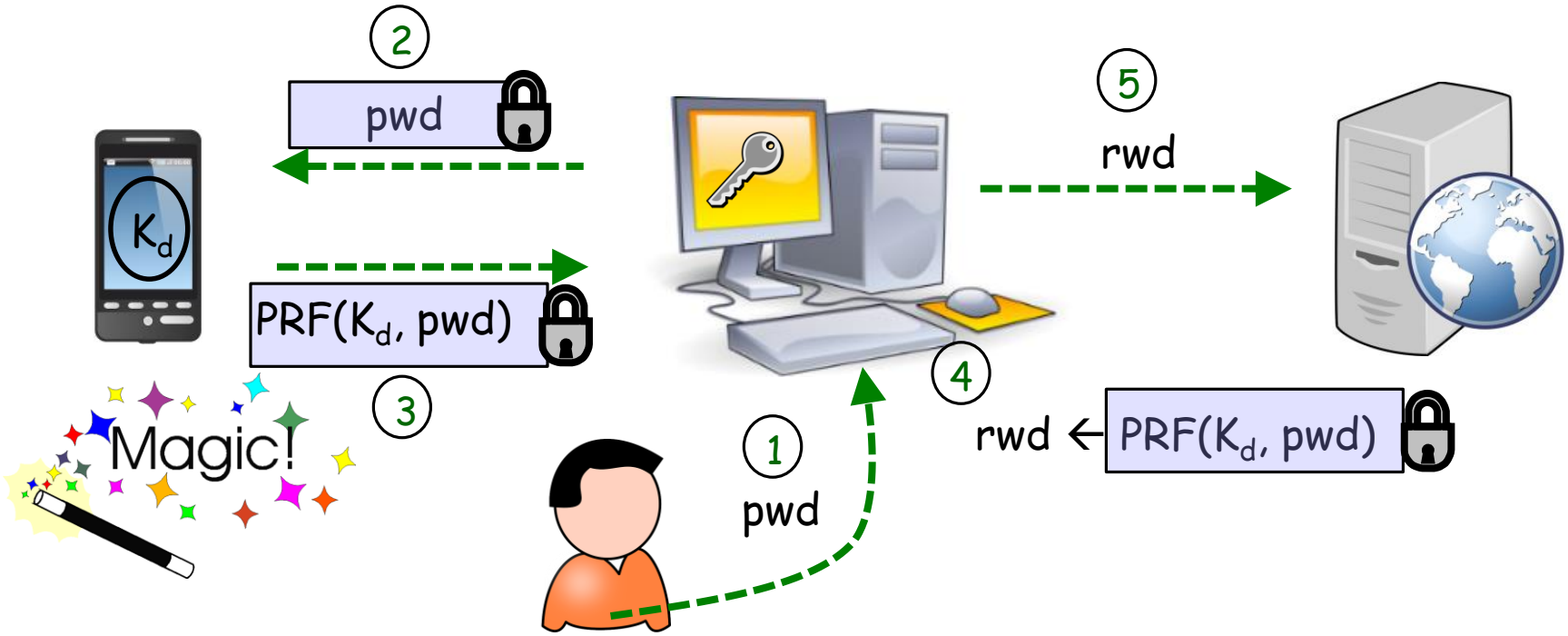


☺ rwd is a (pseudo) random password → *offline attacks are infeasible*

☺ Storage in device (K_d) is *independent* of master pwd and of rwd's

☹ Master pwd is sent unprotected to device

Oblivious PRF ← OPRF-based Solution

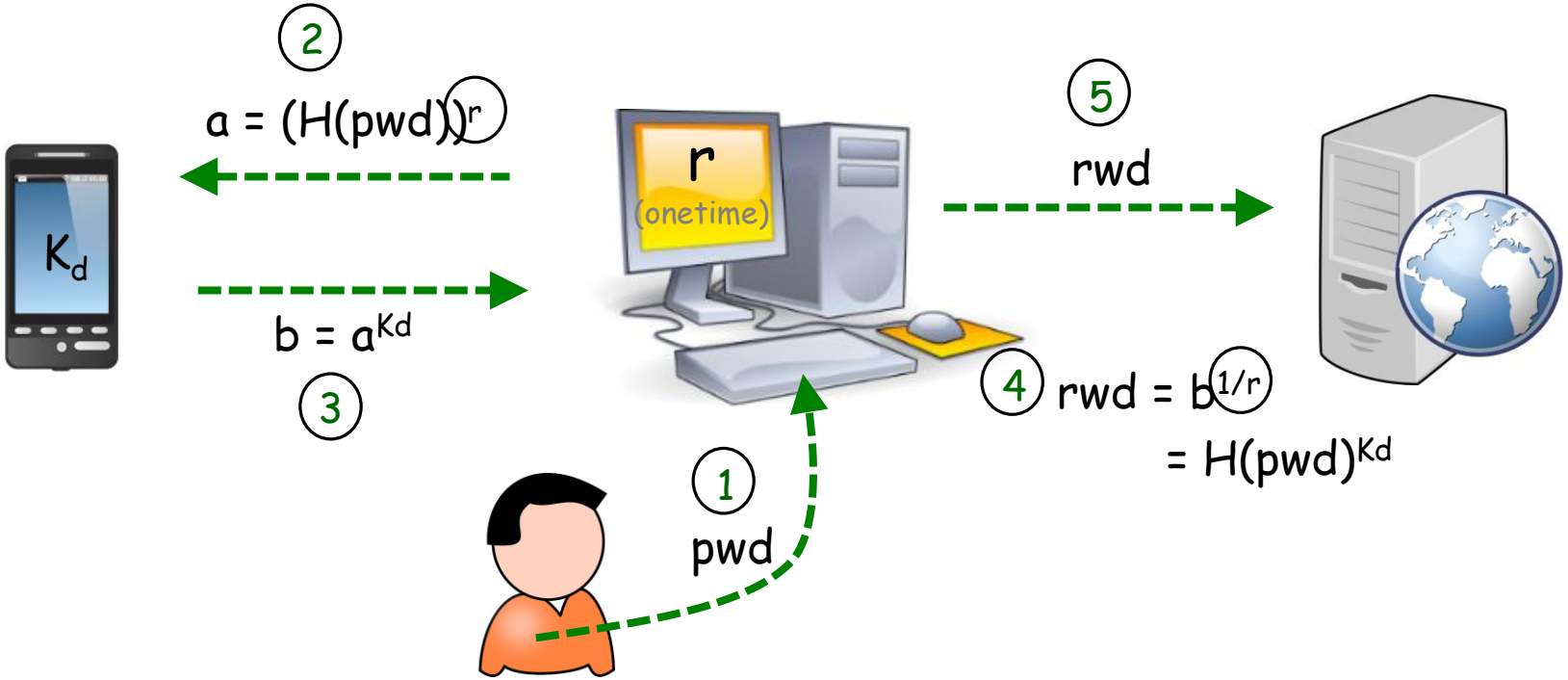


☺ rwd is a (pseudo) random password → *offline attacks are infeasible*

☺ Storage in device is *independent* of master pwd and of individual rwd's

☺ **Master pwd hidden over the wire and from the device!**

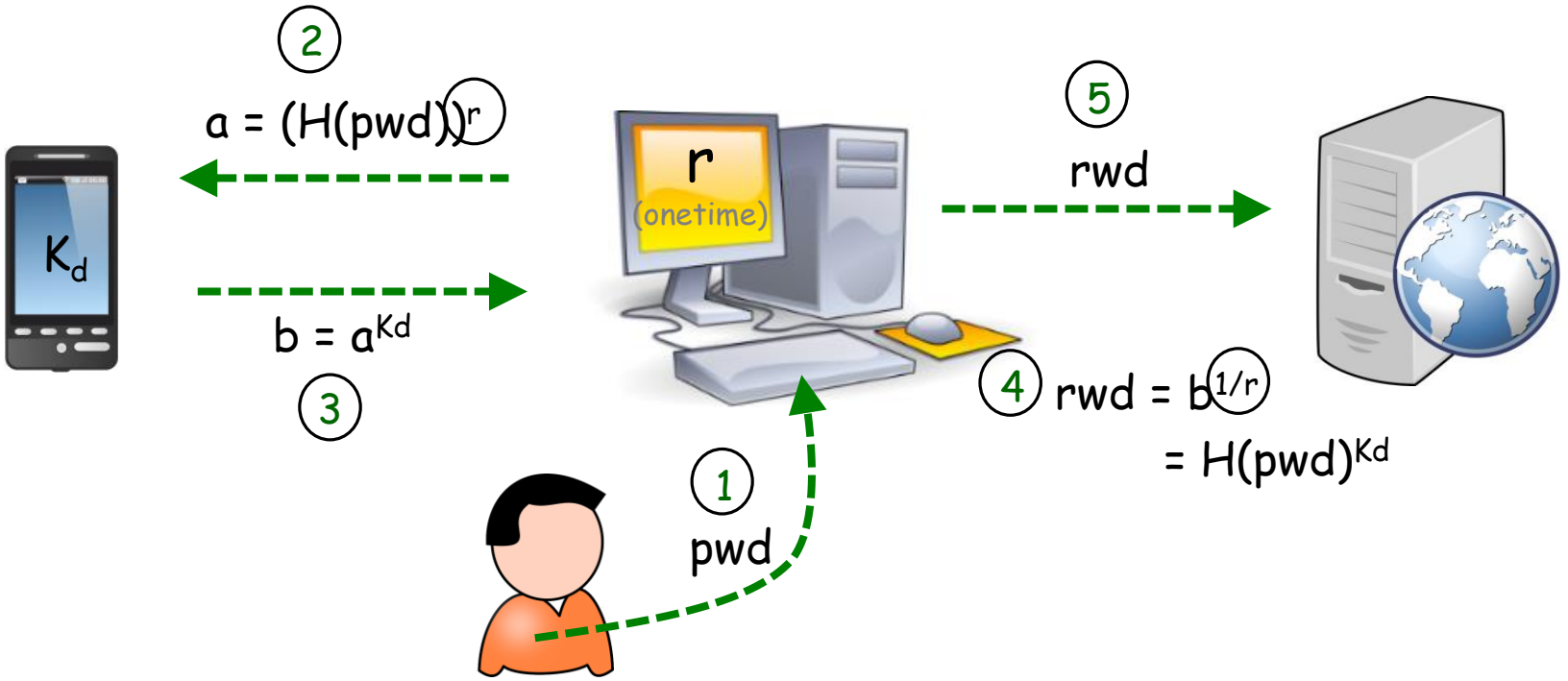
Implementation: $OPRF(K_d, pwd) = (H(pwd))^{K_d}$



☺ ☺ rwd is pseudo-random and K_d fully independent of pwd and of rwd

☺ ☺ *master pwd is **perfectly** hidden on the wire and from device!!*

Implementation: $OPRF(K_d, pwd) = (H(pwd))^{K_d}$



☺ ☺ rwd is pseudo-random and K_d fully indepen… …wd

CAN WORK WITH ANY CLIENT-SERVER PASSWORD PROTOCOL (SERVER TRANSPARENT)

☺ ☺ …*…the wire and from device!!*

Correction

- The next slide contains a correction with respect to the RWC talk. It notes that if an attacker can learn the plaintext rwd upon server compromise (not possible for PAKE protocols but possible for the standard password-over-TLS) the device should authenticate to the client (but client-to-device authentication is not required).

Not only secure…

- **Performance:** Single round C-D, 1 exponentiation for D, 2 for C, and one hash into group for C (any DH group works, no bilinear, etc)
  - SPHINX pwd manager: Implementation as Android app + usability study (user only inputs master pwd, rest is automated) – see references
- **Server transparent** (works with Google, Facebook, your employer…)
  - No need to protect against an eavesdropper (self-protected by SPHINX) or to authenticate user/client to device
  - Requires device authentication if attacker can find plaintext rwd upon server compromise (possible with password-over-tls but not for PAKE)
- ***Can replace D with online service***
  - pwd, rwd never seen by server; server needs to authenticate to client; client-to-server authentication not needed

SPHINX Security

- Device compromise: *Unconditional* secure pwd/rwd (online-only att'ck)
- Network attacks: *Unconditional* security device-client communication
  - *No PKI or externally enforced secure channels* (great for **online** SPHINX)
- Offline dictionary attacks: *Infeasible* (random rwd)
  - Offline against master pwd ONLY if *both server and device compromised*
- Online dictionary attacks: *Infeasible* (random and independent rwd's)
- Password leakage: Partial defense (rwd useless in another server, master pwd useless w/o device, url hashing prevents phishing)

Two-factor authentication with improved security and usability (see ref's)

Part II:

How to Protect* a *Valuable** Secret*

When all You Remember is a Password

*** Protect: Secrecy and Availability**

****** Bitcoin wallet, user-controlled cloud backup, secure msging keys, private key for a PK credential, corporate keys,…

How to store a secret

- Protect *secrecy* and *availability* of information while remembering a *single* password
  - Need a multi-server solution
  - Single server → Single point of failure for secrecy (offline dict attacks) and availability (server gone → secret gone).
- Natural cryptographic solution: keep the secret encrypted in multiple locations; *secret share the encryption key* in multiple servers
  - Share among n servers, retrieve from t+1 servers (e.g. n=5, t=2)
- Protects *availability and secrecy*:
  - Available: As long as t+1 available
  - Secret: As long as no more than t corrupted

Wait, but how do you authenticate to each server for share retrieval?

- Server needs to authenticate the user before delivering a share
- All we have is a user and a password
  - A strong independent password with each server? Not realistic
  - Same (or slight-variant) password for each server? Not good

➔ *Each server as a single point of failure!*

  - From one point of failure to n. We didn't achieve much, did we?

What we really want: PPSS [BJSL'12] (Password Protected Secret Sharing)

- Init: User secret shares a secret among n servers; **forgets secret** and keeps a **single password**.
- Retrieval: User contacts t + 1 servers, authenticates using the **single password** and **reconstructs the secret**.
- Security: Attacker that breaks into t servers learns nothing about secret or password
  - Even if it and finds all the server's secret information (shares, long-term keys, password file, etc.)
  - Only adversary option: Guess the password, try it in an online attack.
  - Offline attacks with ≤ t corrupted servers are useless.

+ **Soundness**: User reconstructs the correct secret or else rejects.

A surprisingly efficient PPSS scheme

- Computation:
  - Single exponentiation for each server
  - Only two exponentiations *in total* for the client (independent of t and n)
  - t multiplications (additions in ECC) for client and for each server
- Communication: Single parallel message from user to t+1 servers, one message back from each server. No inter-server communication.
- *No assumed PKI or secure channels* (other than for initialization)
- Any t, n ($t \leq n$)
- See papers
- Note: Extension to Threshold-PAKE.

Part III: X-PAKE*

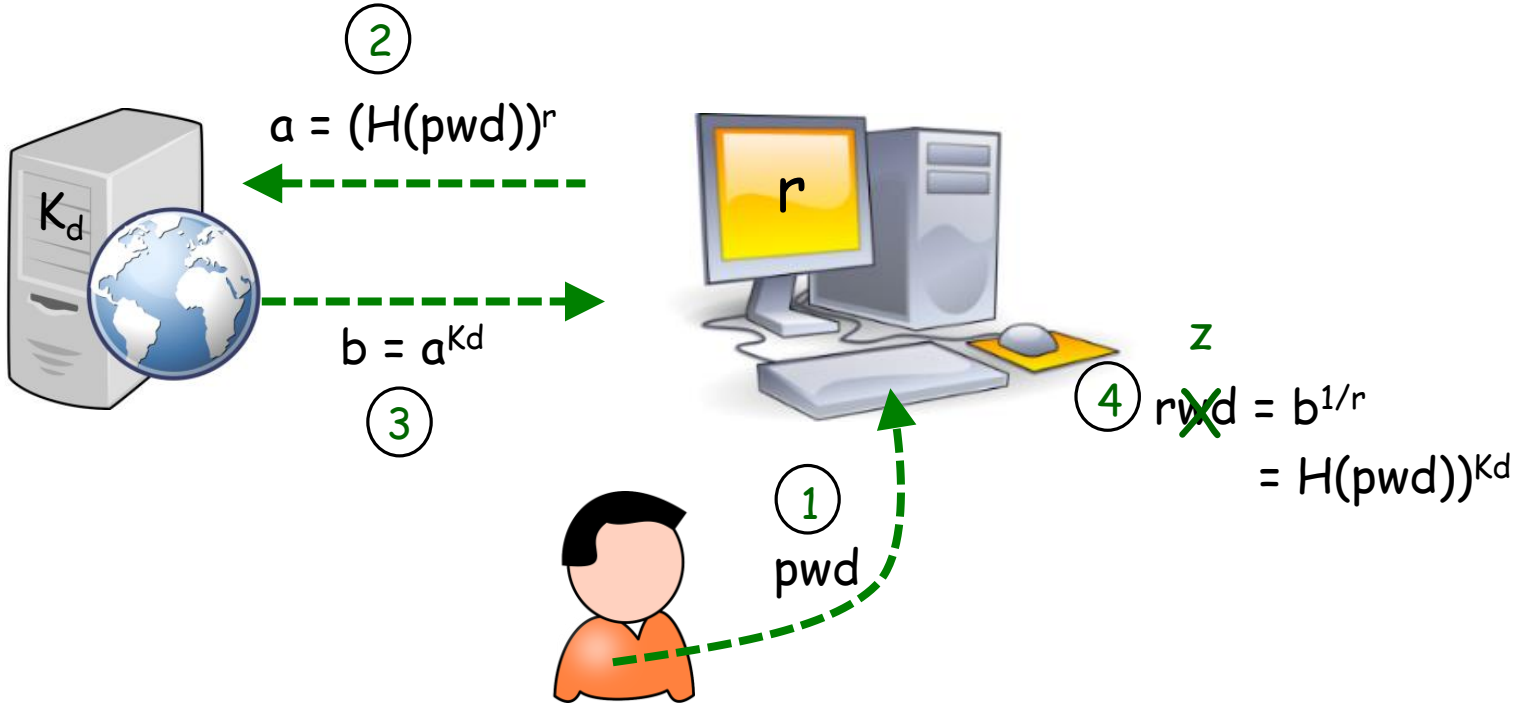
Enhanced Password Security for the Single-Server Setting (without PKI)

*Follows from (1,1)-PPSS, ~ Boyen'09

Single-Server PAKE (asymmetric/augmented PAKE, aPAKE)

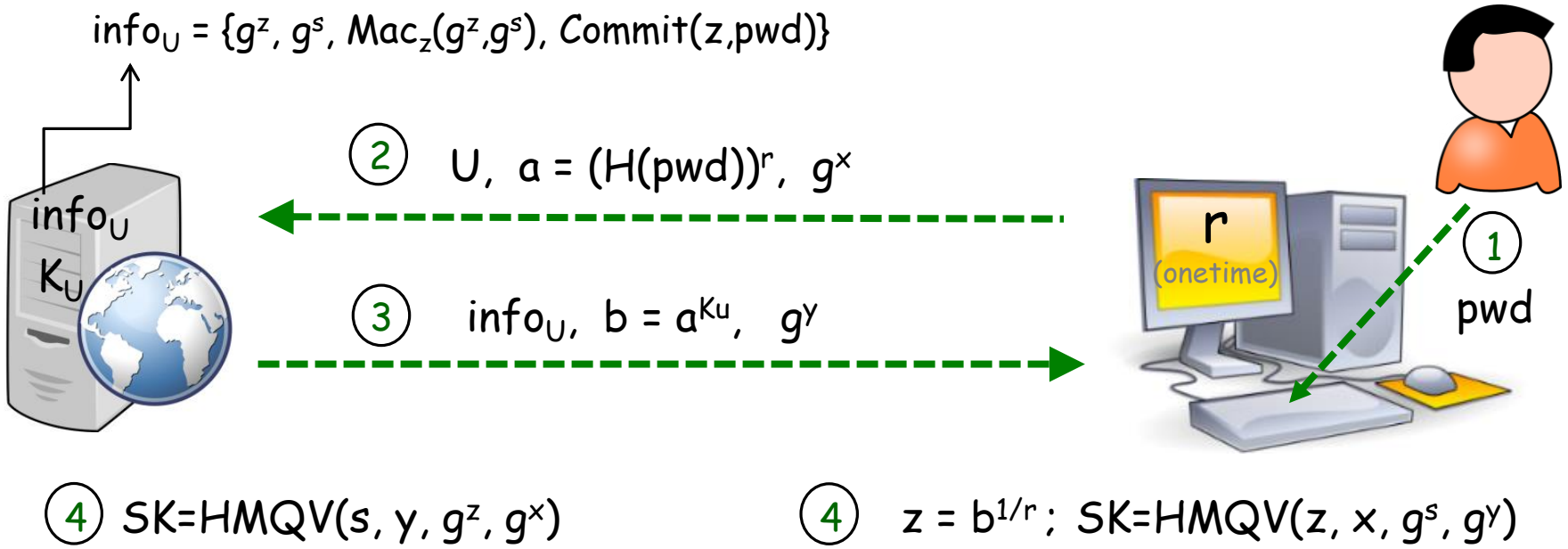
- Desiderata:
  1. Force attacker to run a dictionary attack upon server compromise
  2. No pre-computation prior to server compromise should help
  3. Server should never see the password in plaintext
  4. Reduce/eliminate reliance on PKI
  5. Performance: Offload hash iterations to client ("key stretching")
- Password-over-TLS:  1, 2  3, 4, 5
- PKI-free aPAKE:  1, 3, 4  2, 5
- **X-PAKE:**  **1, 2, 3, 4, 5**

X-PAKE



- z is a secret only U knows; can be used to bootstrap a AKE with S
  - E.g., S stores public keys PK_U=PK(z), PK_S=PK(s) which it sends to U
  - U and S use their PK's and private keys (z and s) with any PK-based AKE (note: AKE and mutual authentication)

X-PAKE with HMQV (single round)



- Single round (one message per party, add'l one for explicit auth)
- HMQV complexity (~ plain DH) + 1 exp for S, 2 exp for U

X-PAKE

- X-PAKE: X is for "ECS" (Enhanced Client-Server) PAKE
- **No reliance on PKI**
- Server **never sees password**, *not even at init* (good against pwd reuse)
- **Private salt**: Attacker cannot pre-compute dictionary
  - Afaik, no other PKI-free aPAKE achieves this!
- Hash **iterations can be offloaded** to user [Boyen'09]
  - Afaik, no other aPAKE w/private salt (incl PKI-based) achieves this!
- Hedged PKI: **TLS-protected PAKE vs PAKE-protected TLS**
  (+ hidden pwd + offload iterations)
  - If PKI acceptable can use z as client signature key and TLS client auth

Summary

- Password vulnerabilities: A serious problem endangering everything from our privacy to social well-being to national security.
- Yet, we showed that *password insecurity is not inevitable*.
- Blinded DH to the rescue in three important applications:
  - Password store with perfect security (device-based and/or online)
  - Password protected secret sharing (multi-server secret protection with a single memorized password)
  - X-PAKE: Asymmetric PAKE with extra (ordinary) properties (PKI-free, private salt, client iterated, …)
- All schemes backed by security models and proofs of security

Mature, efficient, simple technology, just waiting to be deployed…

Thanks!

Annotated references:

http://webee.technion.ac.il/~hugo/rwc17.html

http://alturl.com/aaxev