Side-Channels Attacks on PAKE protocols

Daniel De Almeida Braga

CEA - October, 11th 2022

Me, Myself and I

Cryptography in the Wild: The Security of Cryptographic Implementations and Standards

What I Have Been Doing

Cryptography in the Wild: The Security of Cryptographic Implementations and Standards

- Smart Cards protocol (SCP10)

Cryptography in the Wild: The Security of Cryptographic Implementations and Standards

- Smart Cards protocol (SCP10)
- Password Authenticated Key Exchange (PAKE)

Cryptography in the Wild: The Security of Cryptographic Implementations and Standards

- Smart Cards protocol (SCP10)
- Password Authenticated Key Exchange (PAKE)
  - Dragonfly (WPA3, EAP-pwd)

Cryptography in the Wild: The Security of Cryptographic Implementations and Standards

- Smart Cards protocol (SCP10)
- Password Authenticated Key Exchange (PAKE)
  - Dragonfly (WPA3, EAP-pwd)
  - SRP (deployed in many projects)

Cryptography in the Wild: The Security of Cryptographic Implementations and Standards

- Smart Cards protocol (SCP10)
- Password Authenticated Key Exchange (PAKE)
  - Dragonfly (WPA3, EAP-pwd)
  - SRP (deployed in many projects)
- User study on constant time tools usage/usability

Cryptography in the Wild: The Security of Cryptographic Implementations and Standards

- Smart Cards protocol (SCP10)
- Password Authenticated Key Exchange (PAKE)
  - Dragonfly (WPA3, EAP-pwd)
  - SRP (deployed in many projects)
- User study on constant time tools usage/usability
- Formally verified implementations and constant-time verification tools

Today's Topics

1. Microarchitectural side-channel attacks on Dragonfly

Today's Topics

1. Microarchitectural side-channel attacks on Dragonfly
   - How does it work?
   - Finding is easier than exploiting

Today's Topics

1. Microarchitectural side-channel attacks on Dragonfly
   - How does it work?
   - Finding is easier than exploiting
2. How could these attacks have been prevented?

Today's Topics

1. Microarchitectural side-channel attacks on Dragonfly
   - How does it work?
   - Finding is easier than exploiting
2. How could these attacks have been prevented?
   - Why were the implementations vulnerable?

Today's Topics

1. Microarchitectural side-channel attacks on Dragonfly
   - How does it work?
   - Finding is easier than exploiting
2. How could these attacks have been prevented?
   - Why were the implementations vulnerable?
3. Are there sustainable ways to fix these vulnerabilities?

Context and Motivations

A Few Words About PAKEs

What to expect from a PAKE, starting from a password:

- Authentication
- End up with a strong key
- Resist to (offline) dictionary attack

A Few Words About PAKEs

What to expect from a PAKE, starting from a password:

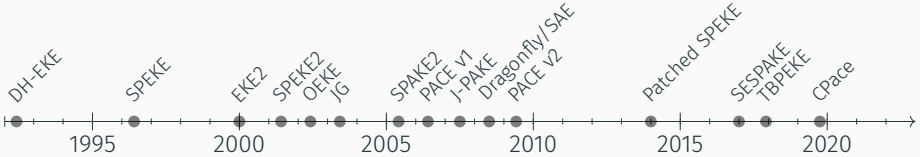
- Authentication
- End up with a strong key
- Resist to (offline) dictionary attack

Lots of different PAKEs

- Balanced/Symetric

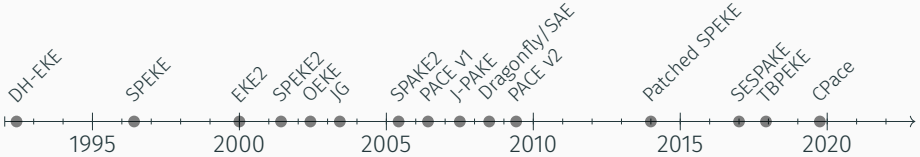
Lots of different PAKEs

- Balanced/Symetric

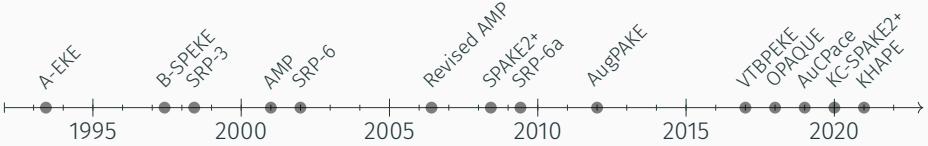


Lots of different PAKEs

- Balanced/Symetric

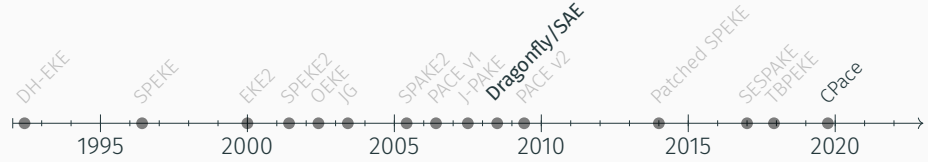


- Augmented/Asymetric

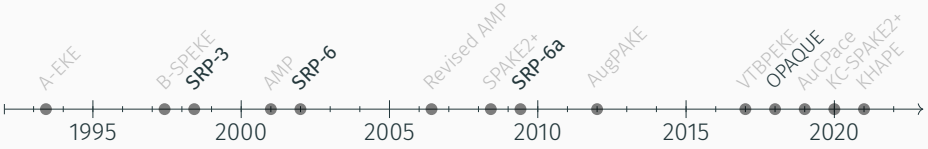


Lots of different PAKEs

- Balanced/Symetric



- Augmented/Asymetric



Why Looking at PAKEs?

Recent interest (WPA3 and standardization) with practical security considerations

Why Looking at PAKEs?

Recent interest (WPA3 and standardization) with practical security considerations

- Dragonfly and WPA3: Dragonblood[1] and attack refinement[2]

[1] M.Vanhoef and E.Ronen *Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd.* In IEEE S&P. 2020
[2] D. De Almeida Braga et al. *Dragonblood Is Still Leaking: Practical Cache-based Side-Channel in the Wild.* In ACSAC. 2020

Why Looking at PAKEs?

Recent interest (WPA3 and standardization) with practical security considerations

- Dragonfly and WPA3: Dragonblood[1] and attack refinement[2]
- Partitioning Oracle Attack[3] applied to some OPAQUE implementations

[1] M.Vanhoef and E.Ronen *Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd.* In IEEE S&P. 2020
[2] D. De Almeida Braga et al. *Dragonblood Is Still Leaking: Practical Cache-based Side-Channel in the Wild.* In ACSAC. 2020
[3] J.Len et al. *Partitioning Oracle Attack.* In USENIX Security. 2021

Why Looking at PAKEs?

Recent interest (WPA3 and standardization) with practical security considerations

- Dragonfly and WPA3: Dragonblood[1] and attack refinement[2]
- Partitioning Oracle Attack[3] applied to some OPAQUE implementations
- Attacks on SRP[4, 5]

[1] M.Vanhoef and E.Ronen *Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd.* In IEEE S&P. 2020
[2] D. De Almeida Braga et al. *Dragonblood Is Still Leaking: Practical Cache-based Side-Channel in the Wild.* In ACSAC. 2020
[3] J.Len et al. *Partitioning Oracle Attack.* In USENIX Security. 2021
[4] A.Russon *Threat for the Secure Remote Password Protocol and a Leak in Apple's Cryptographic Library.* In ACSN. 2021
[5] D. De Almeida Braga et al. *PARASITE: PAssword Recovery Attack against Srp Implementations in ThE wild.* In ACM CCS. 2021

Why Looking at PAKEs?

Recent interest (WPA3 and standardization) with practical security considerations

- Dragonfly and WPA3: Dragonblood[1] and attack refinement[2]
- Partitioning Oracle Attack[3] applied to some OPAQUE implementations
- Attacks on SRP[4, 5]

Lesson to learn: Small leakage can be devastating

[1] M.Vanhoef and E.Ronen *Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd.* In IEEE S&P. 2020
[2] D. De Almeida Braga et al. *Dragonblood Is Still Leaking: Practical Cache-based Side-Channel in the Wild.* In ACSAC. 2020
[3] J.Len et al. *Partitioning Oracle Attack.* In USENIX Security. 2021
[4] A.Russon *Threat for the Secure Remote Password Protocol and a Leak in Apple's Cryptographic Library.* In ACSN. 2021
[5] D. De Almeida Braga et al. *PARASITE: PAssword Recovery Attack against Srp Implementations in ThE wild.* In ACM CCS. 2021

Side Channel Attacks

```
def processPassword(pwd):
  if "a" in pwd:
     res = long_processing(pwd)
  else:
     res = short_processing(pwd)
   return res
```

Side Channel Attacks

```
def processPassword(pwd):
  if "a" in pwd:
     res = long_processing(pwd)
  else:
     res = short_processing(pwd)
   return res
```

Gain information through timing:


0.5 seconds ⇒ no *a*


10 seconds ⇒ *a*

Side Channel Attacks

```
def processPassword(pwd):
  if "a" in pwd:
     res = long_processing(pwd)
  else:
     res = short_processing(pwd)
   return res
```

```
def processPassword2(pwd):
  if "a" in pwd:
    res = long_processing(pwd)
  else:
    res = long_processing2(pwd)
  return res
```

Gain information through timing:


0.5 seconds $\Rightarrow$ no *a*


10 seconds $\Rightarrow$ *a*

Side Channel Attacks

```
def processPassword(pwd):
  if "a" in pwd:
     res = long_processing(pwd)
  else:
     res = short_processing(pwd)
   return res
```

```
def processPassword2(pwd):
  if "a" in pwd:
    res = long_processing(pwd)
  else:
    res = long_processing2(pwd)
  return res
```

Gain information through timing:

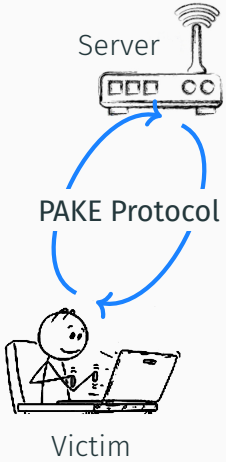
0.5 seconds $\Rightarrow$ no *a*

10 seconds $\Rightarrow$ *a*

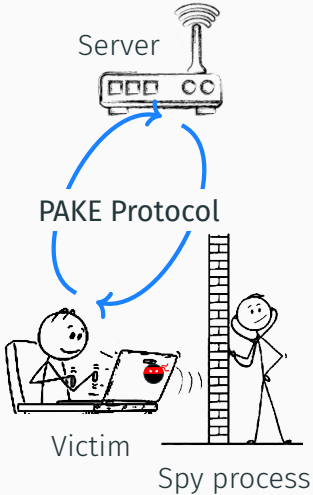
Gain information execution flow:

- Execute `long_processing` $\Rightarrow$ *a*
- Else, no *a* in pwd

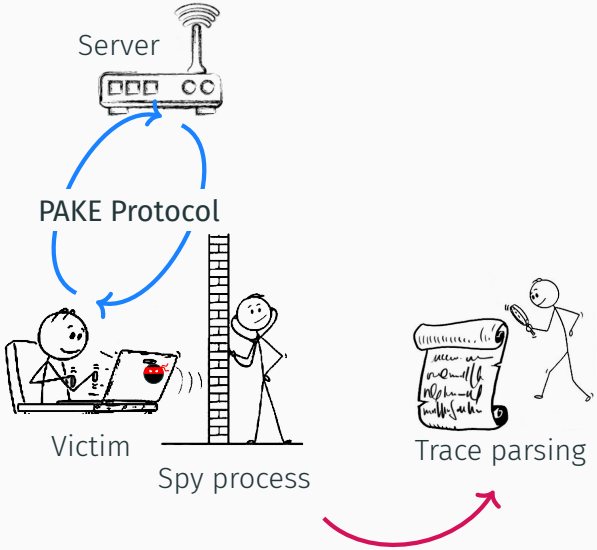
Attack Workflow



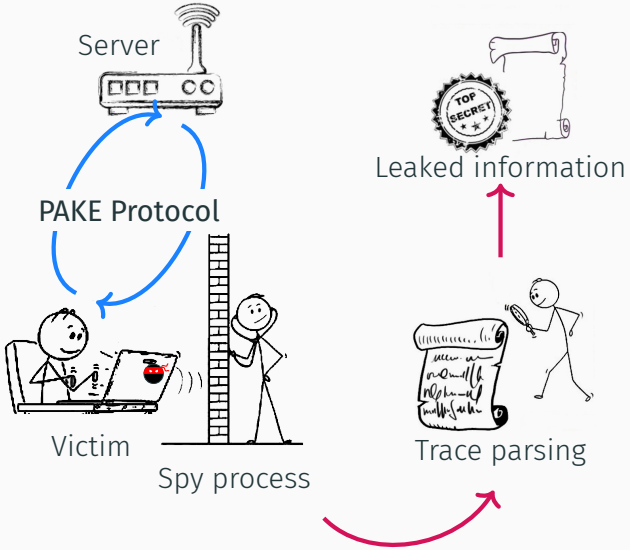
Attack Workflow



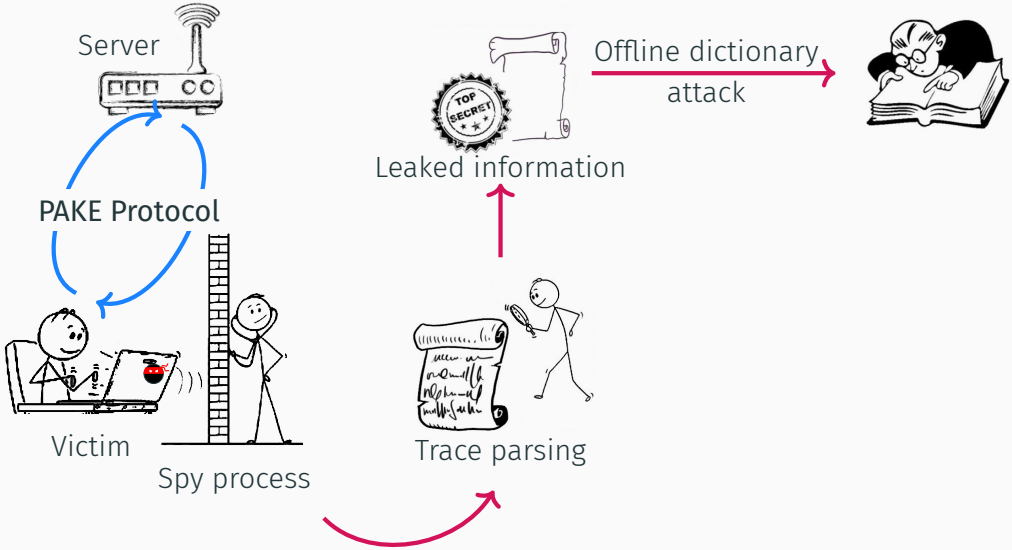
Attack Workflow



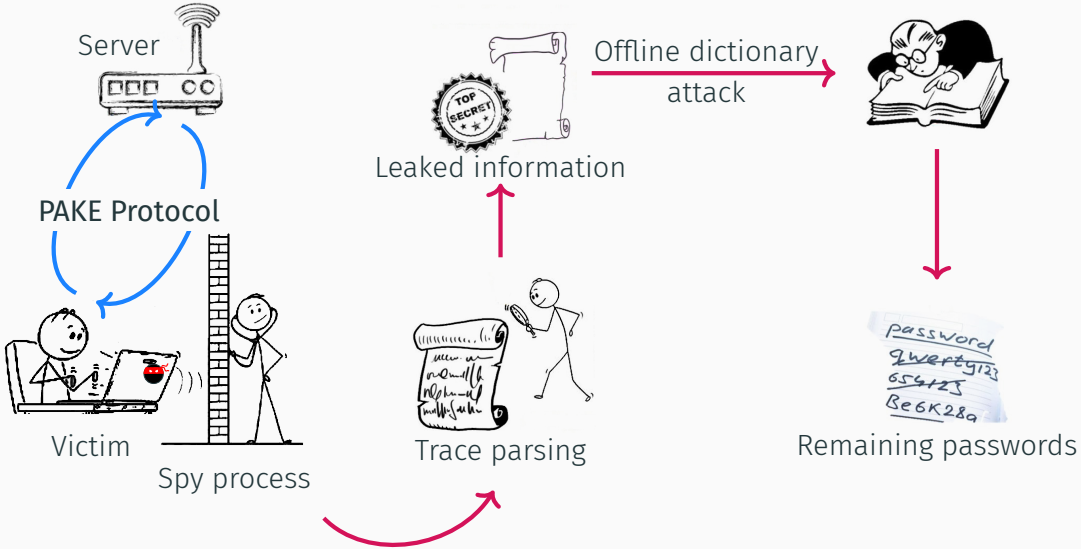
Attack Workflow



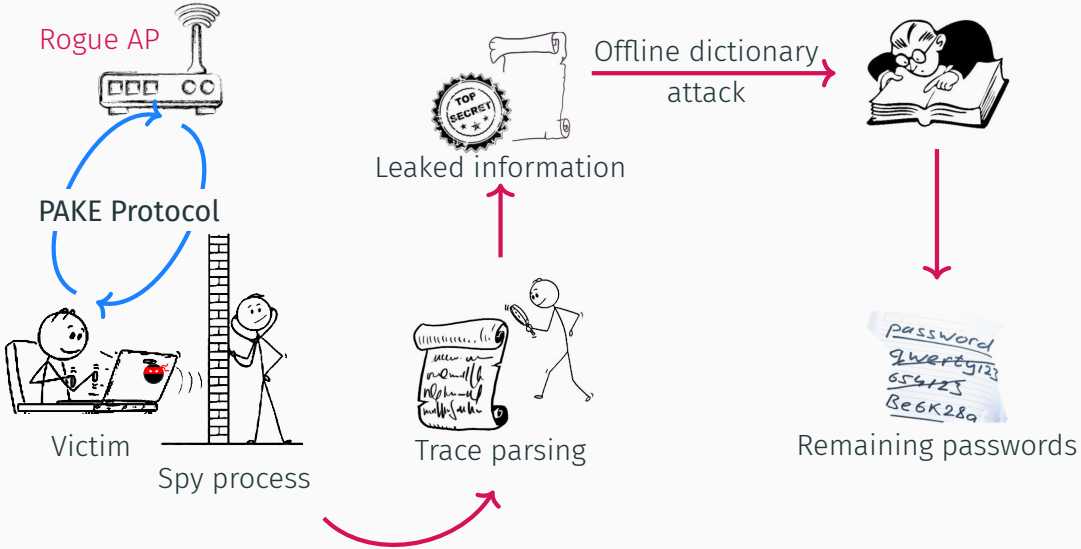
Attack Workflow



Attack Workflow



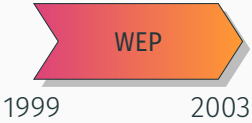
Attack Workflow



Side Channels in Dragonfly/SAE (WPA3)

Toward Secure Wi-Fi Protocols...

Toward Secure Wi-Fi Protocols...



Toward Secure Wi-Fi Protocols...

Toward Secure Wi-Fi Protocols...

Toward Secure Wi-Fi Protocols...

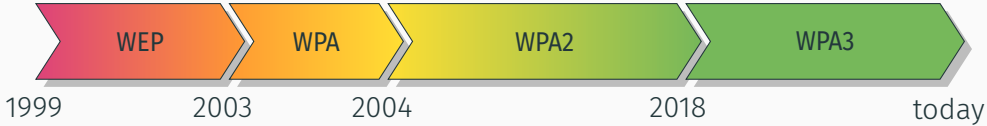
Offline dictionary attack

Toward Secure Wi-Fi Protocols...

Offline dictionary attack

KRACK attack

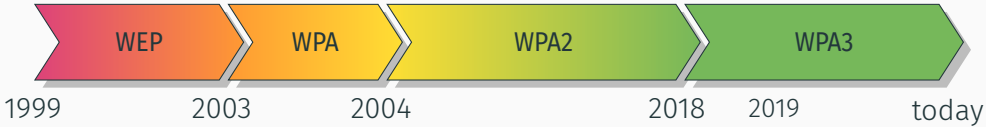
Toward Secure Wi-Fi Protocols...



+ More secure
+ Based on a PAKE (Dragonfly[1])

[1] D. Harkins, 2015, *Dragonfly Key Exchange*, RFC 7664

… But Still not Bullteproof

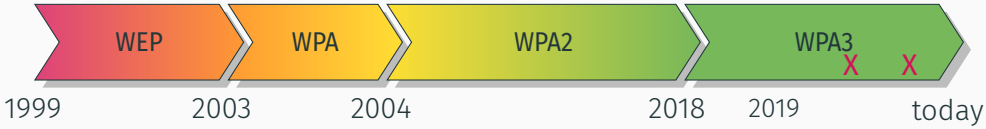


+ More secure
+ Based on a PAKE (Dragonfly)

Dragonblood[1] attacks

[1] M. Vanhoef et al. *Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd*. In IEEE S&P. 2020

… But Still not Bullteproof

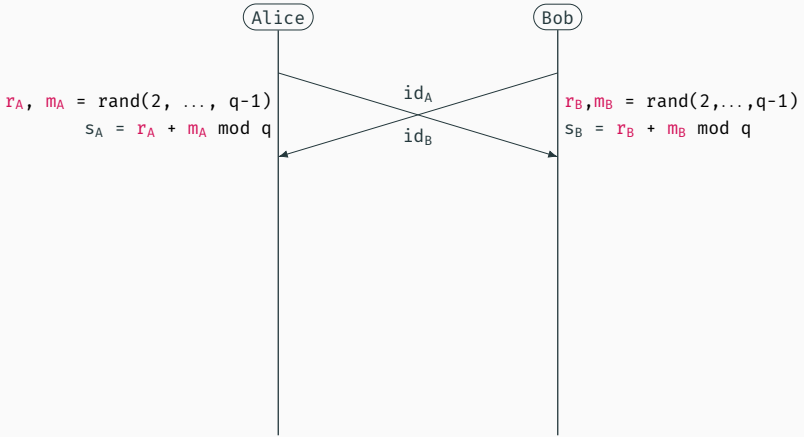


+ More secure
+ Based on a PAKE (Dragonfly)

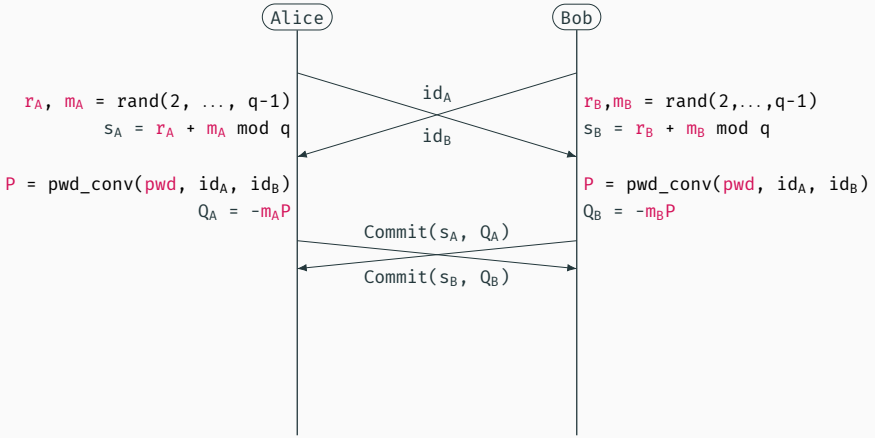
Dragonblood[1] attacks

[1] M. Vanhoef et al. *Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd*. In IEEE S&P. 2020

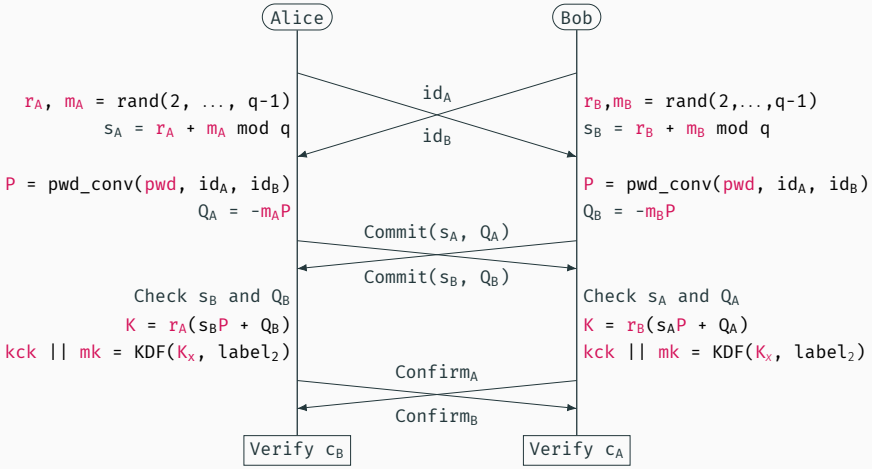
Dragonfly / SAE - A Balanced PAKE



Dragonfly / SAE - A Balanced PAKE



Dragonfly / SAE - A Balanced PAKE



First attack (ACSAC 2020)

A cache-attack that lets us extract

information during the password conversion

leading to an offline dictionary attack.

First attack (ACSAC 2020)

FLUSH+RELOAD[1] and PDA[2]

A cache-attack that lets us extract

information during the password conversion

leading to an offline dictionary attack.

[1] Y. Yarom et al. *Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack.* In USENIX Security Symposium. 2014.
[2] T. Allan et al. *Amplifying side channels through performance degradation.* In ACSAC. 2016

First attack (ACSAC 2020)

FLUSH+RELOAD[1] and PDA[2]

Password to point on an Elliptic Curve

A cache-attack that lets us extract information during the password conversion leading to an offline dictionary attack.

First attack (ACSAC 2020)

FLUSH+RELOAD[1] and PDA[2]

Password to point on an Elliptic Curve

A cache-attack that lets us extract information during the password conversion leading to an offline dictionary attack.

Passive attacker can eliminate wrong passwords from a list

SAE - Probabilistic Password Conversion (EC)

```
def HuntingAndPecking(pwd, MAC_A, MAC_B, k=40):
```

Hash pwd, MAC_A, MAC_B and a counter until we find a point coordinate. Do 40 iterations anyway, but save the first conversion

```
    y = set_coordinates(x, seed_x)
    return (x, y)
```

SAE - Probabilistic Password Conversion (EC)

```
def HuntingAndPecking(pwd, MAC_A, MAC_B, k=40):
  found, i = false, 1
  while not found or i < k:
    seed = Hash(MAC_A, MAC_B, pwd, i)
    x_cand = KDF(seed, label)
    if x_cand is a point's coordinate:
      if not found:
        found, x, seed_x = true, x_cand, seed
        pwd = get_random()
      i = i + 1
  y = set_coordinates(x, seed_x)
  return (x, y)
```

SAE - Probabilistic Password Conversion (EC)

```
def HuntingAndPecking(pwd, MAC_A, MAC_B, k=40):
  found, i = false, 1
  while not found or i < k:
    seed = Hash(MAC_A, MAC_B, pwd, i)
    x_cand = KDF(seed, label)
    if x_cand is a point's coordinate:
      if not found:
        found, x, seed_x = true, x_cand, seed
        pwd = get_random()
      i = i + 1
  y = set_coordinates(x, seed_x)
  return (x, y)
```

← 🕵: new iteration

SAE - Probabilistic Password Conversion (EC)

```
def HuntingAndPecking(pwd, MAC_A, MAC_B, k=40):
  found, i = false, 1
  while not found or i < k:
    seed = Hash(MAC_A, MAC_B, pwd, i)
    x_cand = KDF(seed, label)
    if x_cand is a point's coordinate:
      if not found:
        found, x, seed_x = true, x_cand, seed
        pwd = get_random()
      i = i + 1
  y = set_coordinates(x, seed_x)
  return (x, y)
```

← 🥷: new iteration

← 🥷 : successful conversion

SAE - Probabilistic Password Conversion (EC)

```
def HuntingAndPecking(pwd, MAC_A, MAC_B, k=40):
  found, i = false, 1
  while not found or i < k:
    seed = Hash(MAC_A, MAC_B, pwd, i)
    x_cand = KDF(seed, label)
    if x_cand is a point's coordinate:
      if not found:
        found, x, seed_x = true, x_cand, seed
        pwd = get_random()
      i = i + 1
  y = set_coordinates(x, seed_x)
  return (x, y)
```

← 🥷: new iteration

→ `mask = get_random()`

`do_blind_verif(x_cand, mask)`

← 🥷 : successful conversion

SAE - Probabilistic Password Conversion (EC)

```
def HuntingAndPecking(pwd, MAC_A, MAC_B, k=40):
  found, i = false, 1
  while not found or i < k:
    seed = Hash(MAC_A, MAC_B, pwd, i)
    x_cand = KDF(seed, label)
    if x_cand is a point's coordinate:
      if not found:
        found, x, seed_x = true, x_cand, seed
        pwd = get_random()
      i = i + 1
  y = set_coordinates(x, seed_x)
  return (x, y)
```

← 🥷: new iteration

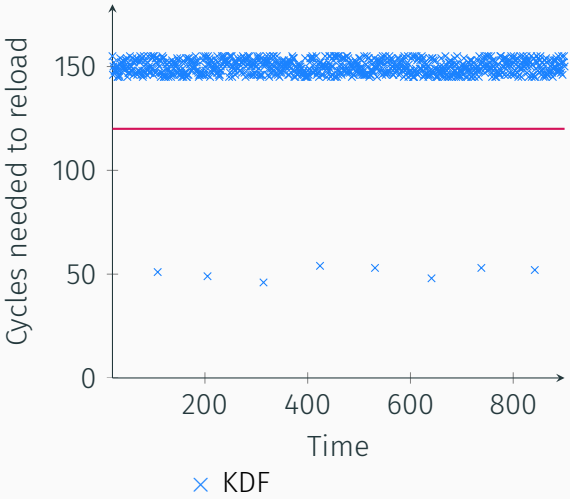
→ mask = get_random()

do_blind_verif(x_{cand}, mask)

← 🥷 : successful conversion

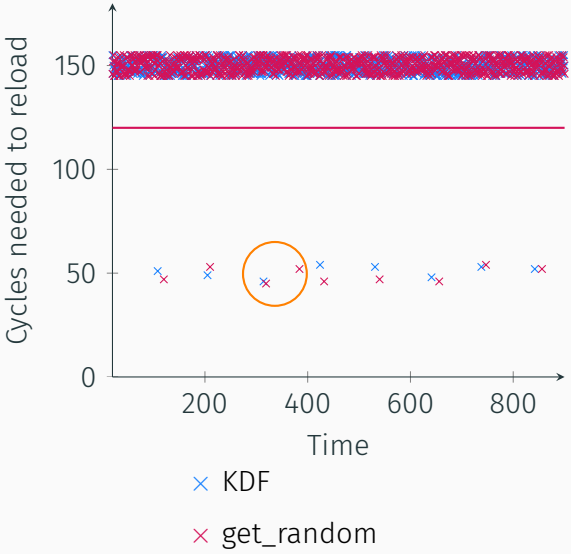
Now What?

- Iteration are easy to distinguish



Now What?

- Iteration are easy to distinguish
- We can guess which iteration is converting the password



Dictionary Reduction

	Iter. for MAC_A, MAC_{B_1}
Leakage	3
pwd_1	
pwd_2	
pwd_3	
pwd_4	
...	
pwd_5	

Dictionary Reduction

	Iter. for MAC_A, MAC_{B_1}
Leakage	3
pwd_1	1
pwd_2	3
pwd_3	3
pwd_4	4
...	...
pwd_5	3

Dictionary Reduction

	Iter. for MAC_A, MAC_{B_1}
Leakage	3
pwd_1	1
pwd_2	3
pwd_3	3
pwd_4	4
...	...
pwd_5	3

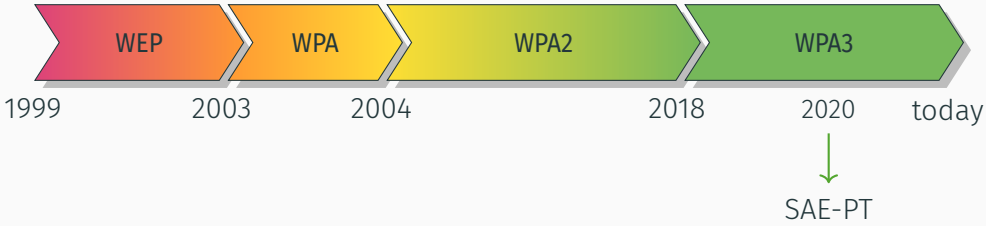
Dictionary Reduction

	Iter. for MAC_A, MAC_{B_1}	Iter. for MAC_A, MAC_{B_2}
Leakage	3	2
pwd_1	1	X
pwd_2	3	8
pwd_3	3	2
pwd_4	4	X
...	...	...
pwd_5	3	1

Dictionary Reduction

	Iter. for MAC_A, MAC_{B_1}	Iter. for MAC_A, MAC_{B_2}
Leakage	3	2
pwd_1	1	X
pwd_2	3	8
pwd_3	3	2
pwd_4	4	X
...	...	...
pwd_5	3	1

Improving the password conversion



Looking under the hood

We mostly analyzed Wi-Fi daemons...

... what about their dependencies, like crypto libraries?

Looking under the hood

We mostly analyzed Wi-Fi daemons...
... what about their dependencies, like crypto libraries?

```
def set_point_coordinate(x, fmt, ec):
```

Looking under the hood

We mostly analyzed Wi-Fi daemons...
... what about their dependencies, like crypto libraries?

```
def set_point_coordinate(x, fmt, ec):
   y = compute_y(x, ec)
   if y_sqr = fmt mod 2:
     y = ec.p - y
   P = create_point(x, y, ec)
   return P
```

Looking under the hood

We mostly analyzed Wi-Fi daemons...
... what about their dependencies, like crypto libraries?

```
def set_point_coordinate(x, fmt, ec):
   y = compute_y(x, ec)
   if y_sqr = fmt mod 2:
     y = ec.p - y
   P = create_point(x, y, ec)
   return P
```

Looking under the hood

We mostly analyzed Wi-Fi daemons...
... what about their dependencies, like crypto libraries?

```
def set_point_coordinate(x, fmt, ec):
   y = compute_y(x, ec)
   if y_sqr = fmt mod 2:
     y = ec.p - y
   P = create_point(x, y, ec)
   return P
```

```
def bin2bn(buf, n):
   # Skip leading 0's
   while (buf[0] == 0):
     n--
     buf++
   [...]
```

Looking under the hood

We mostly analyzed Wi-Fi daemons...
... what about their dependencies, like crypto libraries?

```
def set_point_coordinate(x, fmt, ec):
   y = compute_y(x, ec)
   if y_sqr = fmt mod 2:
     y = ec.p - y
   P = create_point(x, y, ec)
   return P
```

```
def bin2bn(buf, n):
   # Skip leading 0's
   while (buf[0] == 0):
     n--
     buf++
   [...]
```

Looking under the hood

We mostly analyzed Wi-Fi daemons...
... what about their dependencies, like crypto libraries?

```
def set_point_coordinate(x, fmt, ec):
  y = compute_y(x, ec)
  if y_sqr = fmt mod 2:
    y = ec.p - y
  P = create_point(x, y, ec)
  return P
```

```
def bin2bn(buf, n):
  # Skip leading 0's
  while (buf[0] == 0):
    n--
    buf++
  [...]
```

We should have caught this in the first analysis!

Computer Aided Cryptography

Multiple areas..

- Design level
  - Protocol verification (symbolic / computational model)
- Functional correctness / efficiency
  - Correctness, memory safety, ...
- Implementation security

Computer Aided Cryptography

Multiple areas..

- Design level
  - Protocol verification (symbolic / computational model)
- Functional correctness / efficiency
  - Correctness, memory safety, ...
- Implementation security

... with various approaches

- Verifying existing implementations
  - Source code level? Binary level?
  - Leakage model?

Computer Aided Cryptography

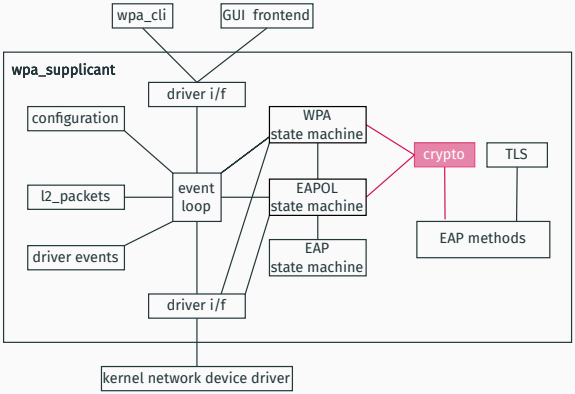
Multiple areas..

- Design level
  - Protocol verification (symbolic / computational model)
- Functional correctness / efficiency
  - Correctness, memory safety, ...
- Implementation security

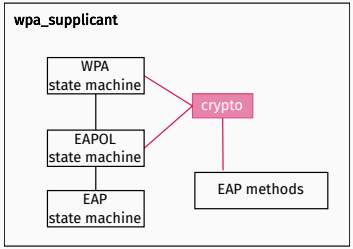
... with various approaches

- Verifying existing implementations
  - Source code level? Binary level?
  - Leakage model?
- Generating formally verified binaries

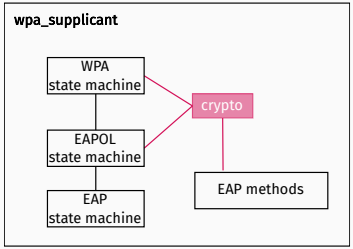
Fixing hostap



Fixing hostap

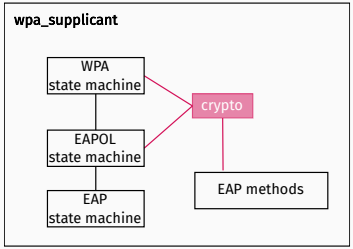


Fixing hostap



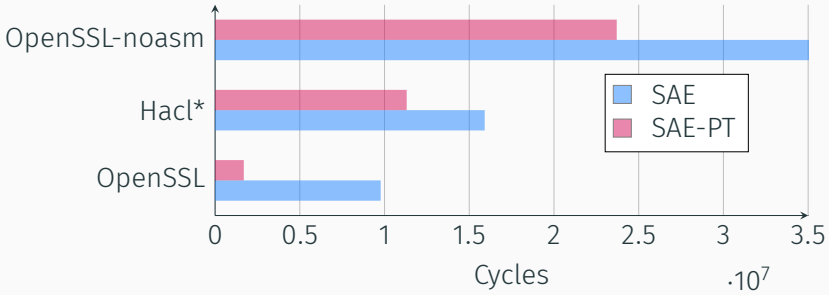

```
crypto/
  ...
  crypto.h

  crypto_mbedtls.c
  crypto_openssl.c
  crypto_wolfssl.c
  ...
```

Fixing hostap




```
crypto/
  ...
  crypto.h
  crypto_hacl.c
  crypto_mbedtls.c
  crypto_openssl.c
  crypto_wolfssl.c
  ...
```

Benchmarks


OpenSSL-noasm
Hacl*
OpenSSL
SAE
SAE-PT
0
0.5
1
1.5
2
2.5
3
3.5
Cycles
$\cdot 10^7$

Constant-time Tools & Usability

Many Tools, Presumably Low Adoption

Tool	Target	Technique
ABPV13	C	Formal
Binsec/Rel	Binary	Symbolic
Blazer	Java	Formal
BPT17	C	Symbolic
CacheAudit	Binary	Formal
CacheD	Trace	Symbolic
COCO-CHANNEL	Java	Symbolic
ctgrind	Binary	Dynamic
ct-fuzz	LLVM	Dynamic
ct-verif	LLVM	Formal
CT-WASM	WASM	Formal
DATA	Binary	Dynamic
dudect	Binary	Statistics

Tool	Target	Technique
FaCT	DSL	Formal
FlowTracker	LLVM	Formal
haybale-pitchfork	LLVM	Symbolic
KMO12	Binary	Formal
MemSan	LLVM	Dynamic
MicroWalk	Binary	Dynamic
SC-Eliminator	LLVM	Formal
SideTrail	LLVM	Formal
Themis	Java	Formal
timecop	Binary	Dynamic
tis-ct	C	Symbolic
VirtualCert	x86	Formal

Why are timing attacks still around?

Are timing attacks part of the threat models of libraries?

- Are developers aware of the threat?
- Do they claim resistance against it?

Why are timing attacks still around?

Are timing attacks part of the threat models of libraries?

- Are developers aware of the threat?
- Do they claim resistance against it?

How do they protect against such attacks?

- Are they tested/verified? If so, how often?

Why are timing attacks still around?

Are timing attacks part of the threat models of libraries?

- Are developers aware of the threat?
- Do they claim resistance against it?

How do they protect against such attacks?

- Are they tested/verified? If so, how often?

Are developers aware of the tools?

- If so, which ones?
- Are they more prone to use a specific tool ”type”?

Why are timing attacks still around?

Are timing attacks part of the threat models of libraries?

- Are developers aware of the threat?
- Do they claim resistance against it?

How do they protect against such attacks?

- Are they tested/verified? If so, how often?

Are developers aware of the tools?

- If so, which ones?
- Are they more prone to use a specific tool ”type”?

Why do we still find textbook issues?

- Are they waiting for specific features?

Let’s ask them!

“They’re not that hard to mitigate”: What Cryptographic Library Developers Think About Timing Attacks

Jan Jancar[1], Marcel Fourné[2], Daniel De Almeida Braga[3], Mohamed Sabt[3], Peter Schwabe[2], Gilles Barthe[2], Pierre-Alain Fouque[3] and Yasemin Acar[2,4]

CROCS Centre for Research on Cryptography and Security [1]

MAX PLANCK INSTITUTE FOR SECURITY AND PRIVACY [2]

UMR IRISA [3]

THE GEORGE WASHINGTON UNIVERSITY [4]
WASHINGTON, DC

Survey

27 librairies

OpenSSL, BearSSL, libgcrypt, s2n (Amazon), RustCrypto, ...

44 valid responses

Various roles (developers, maintainers, committers, ...)

Survey

1. Participant background

27 librairies

OpenSSL, BearSSL, libgcrypt, s2n (Amazon), RustCrypto, ...

44 valid responses

Various roles (developers, maintainers, committers, ...)

Survey

1. Participant background
↓
2. Library properties & decisions

27 librairies

OpenSSL, BearSSL, libgcrypt, s2n (Amazon), RustCrypto, ...

44 valid responses

Various roles (developers, maintainers, committers, ...)

Survey

1. Participant background
↓
2. Library properties & decisions
↓
3. Tool awareness

27 librairies

OpenSSL, BearSSL, libgcrypt, s2n (Amazon), RustCrypto, ...

44 valid responses

Various roles (developers, maintainers, committers, ...)

Survey

1. Participant background
↓
2. Library properties & decisions
↓
3. Tool awareness
↓
4. Tool use

27 librairies

OpenSSL, BearSSL, libgcrypt, s2n (Amazon), RustCrypto, ...

44 valid responses

Various roles (developers, maintainers, committers, ...)

Survey

1. Participant background
↓
2. Library properties & decisions
↓
3. Tool awareness
↓
4. Tool use
↓
5. Hypothetical tool use

27 librairies

OpenSSL, BearSSL, libgcrypt, s2n (Amazon), RustCrypto, ...

44 valid responses

Various roles (developers, maintainers, committers, ...)

Survey

1. Participant background
2. Library properties & decisions
3. Tool awareness
4. Tool use
5. Hypothetical tool use
6. Miscellaneous

27 librairies

OpenSSL, BearSSL, libgcrypt, s2n (Amazon), RustCrypto, ...

44 valid responses

Various roles (developers, maintainers, committers, ...)

Interesting responses

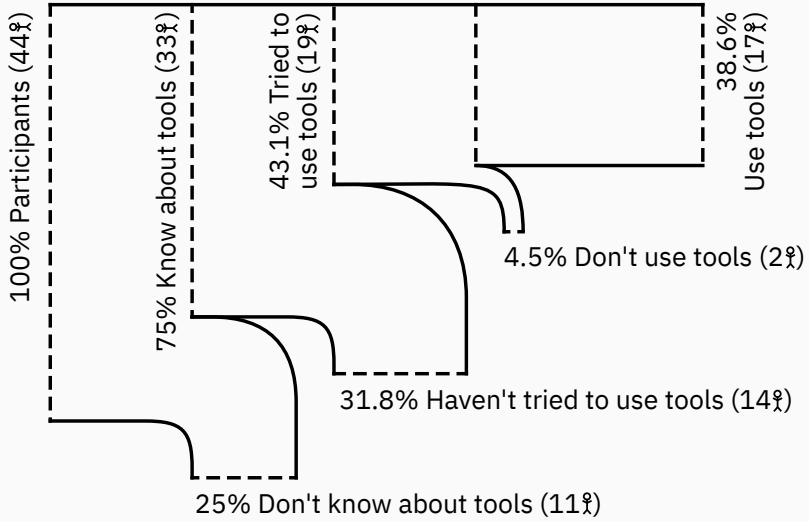
"It was totally obvious for everybody right from the start that protection against timing attacks is ***necessary****."*

"I'm very interested in these sorts of tools, but so far it seems formal analysis tools (at least where we've tried to apply it to correctness) are ***not really usable by mere mortals yet****."*

"For many cases there ***aren't enough real world attacks*** *to justify spending time on preventing timing leaks."*

"They're ***not that hard to mitigate****, at least with the compilers I'm using right now."*

Leaky pipeline



Recommendations

Tool developers

- Make usable tools
- Promote them

Crypto developers

- Use the tools
- Annotate your code

Complier writers

- Support secret types
- Give more control to developers

Standardization bodies

- Encourage to use tools and give recommendations
- Require constant-time code

- PAKEs are spreading
- They are particularly prone to side-channel attacks
- Computer-aided cryptography is nice
- We need more usable tools

Thank you for your attention!

https://gitlab.inria.fr/ddealmei/

@ daniel.de-almeida-braga@irisa.fr