Wi-Fi Protected Setup

Friend or Foe?

Tomáš Rosa

http://crypto.hyperlink.cz

Smart Cards & Devices Forum 2013, May 23rd, Prague

Abstract

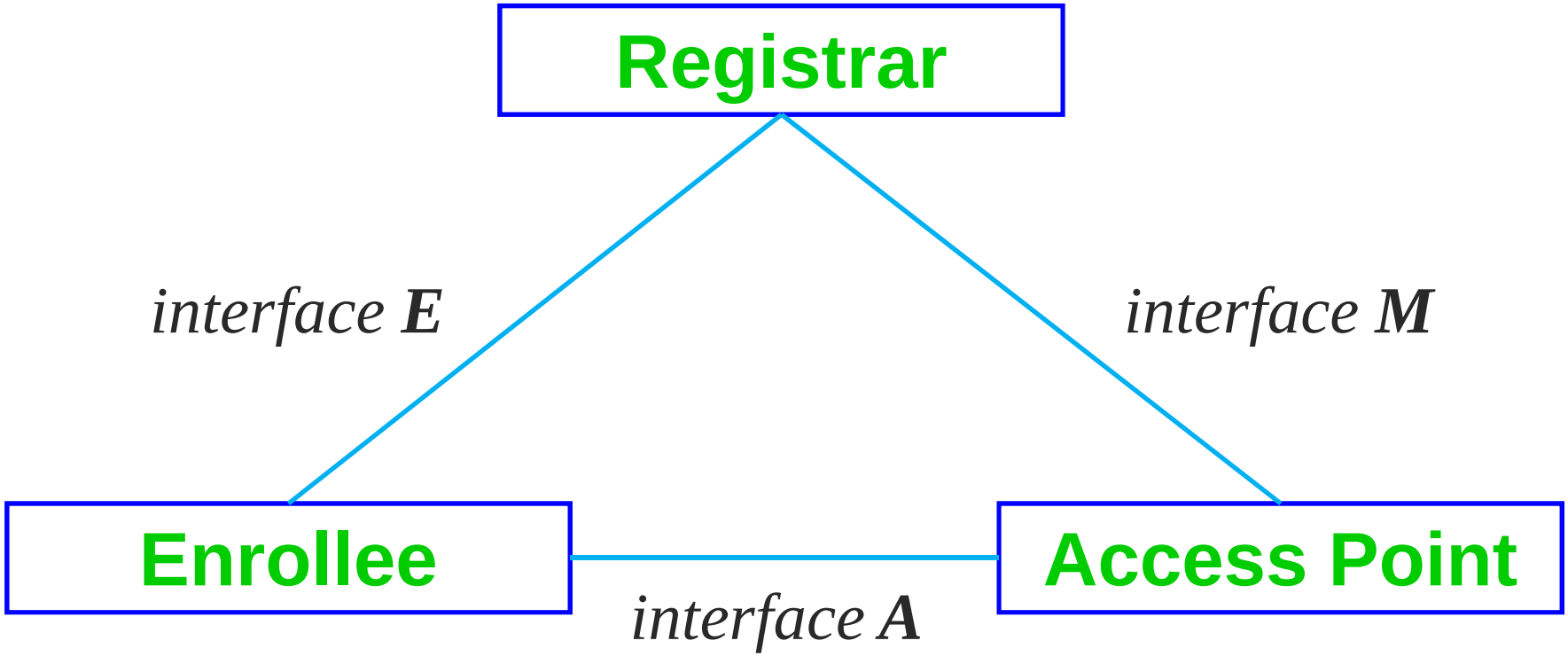
- We review WPS briefly while focusing on selected cryptographic properties of the *Registration Protocol*.
  - This is the core of the whole Wi-Fi Protected Setup.
- We review the known PIN brute force attacks while showing:
  - Where do they occur and how far are they surprising.
  - What else may happen when neglecting standard requirements – so called “dual attack”.
  - How to patch WPS in some other way than following the standard countermeasures *(they have already been there!* [6]*)*

Part ONE
WPS Introduction

WPS Standard Body

- Defined by Wi-Fi Alliance in public, but paid standard [2], [7].
  - It goes in parallel with IEEE 802.11 machinery.
  - It addresses the user experience of automated Wi-Fi setup that is somehow completely neglected by 802.11.
  - Anyway, such activity is crucial for Wi-Fi-based smart devices interworking.
- Also termed as:
  - **“Wi-Fi Simple Configuration” - WSC**

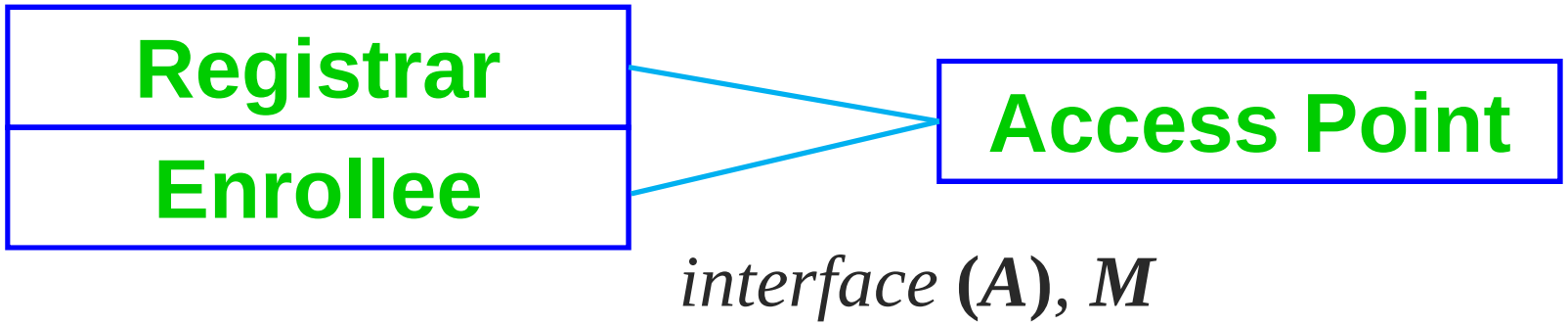
Magic Triangle Tells the Story



Handful of Interfaces

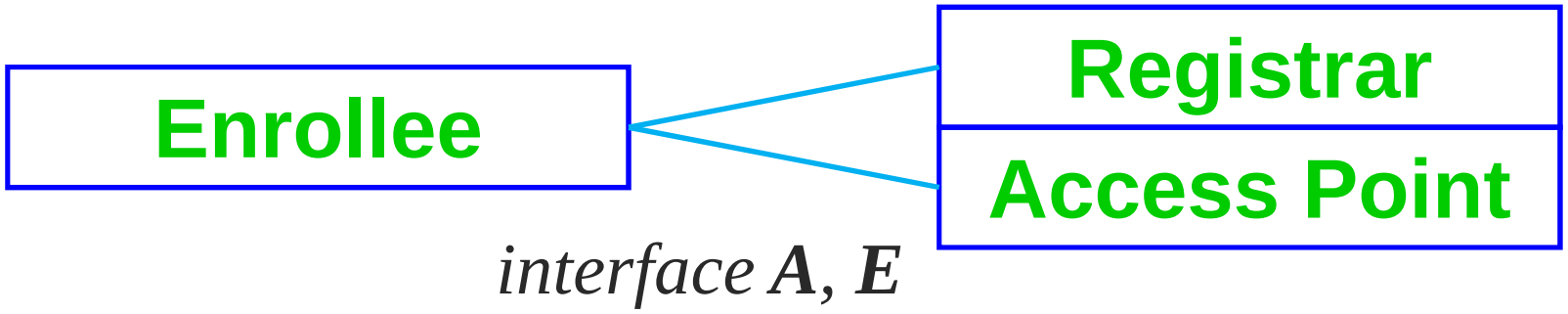
- Interface E
  - *Enables the Registrar to discover and issue WLAN Credentials to the Enrollee.*
  - **EAPOL over 802.11**
- Interface M
  - *Enables an external Registrar to manage WPS AP.*
  - **EAPOL over 802.11**
  - or **UPnP over TCP/IP (over 802.3)**
- Interface A
  - Enables discovery of WPS WLAN and proxies the communication between the Enrollee and IP-only Registrars.
  - **EAPOL over 802.11**

Practical Use Cases I



- Enrollee pretends to be an external Registrar and pulls WLAN credentials from AP for its own setup.
  - After having sipped the secret sauce, the Enrollee resigns its role of WLAN manager.
  - In this setup, the idea of PIN brute force attack originated [5], [9].

Practical Use Cases II

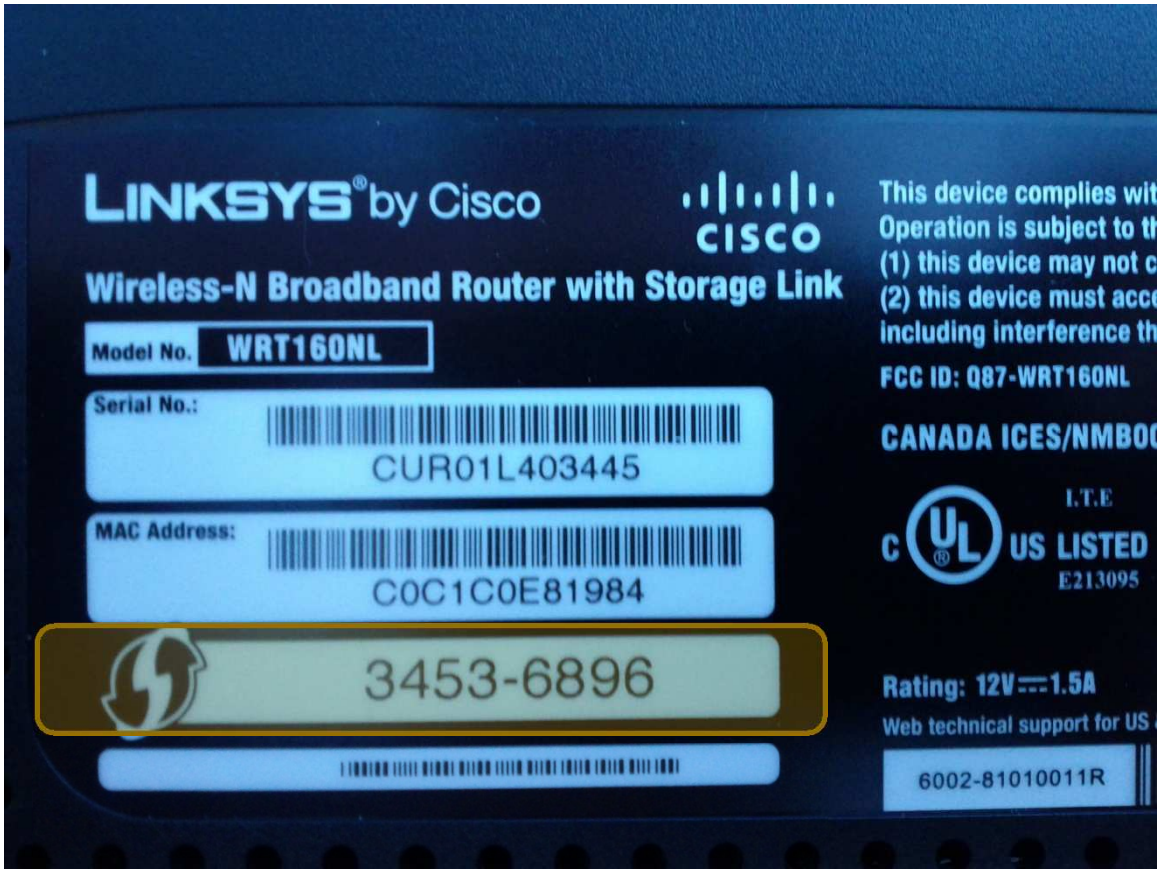


- AP implements built-in Registrar for new Enrollees provisioning.
  - Variants include external UPnP Registrar whose communication with Enrollees is proxied through AP.

Device Password Quest

- In-band mutual authentication for *E* and *M* links is based on Device Password.
  - Registration Protocol describes the core verification procedure.
- To be safe, we shall:
  - Use fresh Device Password for each and every authentication run.
  - Follow special technical procedures for brute-force prevention, etc. [2], [7]

Born Slippy – Static PIN



Oh, Hmm… Check Digit

- Such an 8-digit *static* PIN offers at most 7-digit entropy, since the last digit is a checksum anyway.
  - Public algorithm with no key – as it has to be for interoperability.
  - In particular, here:

$$3*\mathbf{3} + 1*\mathbf{4} + 3*\mathbf{5} + 1*\mathbf{3} + 3*\mathbf{6} + 1*\mathbf{8} + 3*\mathbf{9} + 1*\mathbf{6} = 90 \equiv 0 \pmod{10}$$

Going Simpler Than Simple

- PBC – PushButton Configuration
- Pressing a dedicated button (SW or HW) sets

  Device Password = “0000 0000”
- There are further technical procedures [7] that are worth study.
  - As requirements for secure design.
  - As inspiration for penetration tests.

In Other Words

- What do we need to get a super-strong WPA2 password out of a physically accessible AP?
  - Dismantle the case and start debugging the firmware?
  - Use HW probes for direct memory dump?
  - Invoke complicated side-channel attack?
  - Ask a crystal ball?

Well, Just Push the Button

However…

- This is not to say PBC is terribly wrong concept.
  - Actually, users will probably appreciate this method.
- **This is to remind physical security is often more important than it seems.**

Part TWO
Bit Commitment in Nutshell

Bit Commitment In Nutshell

- Originator announces *C*:

  C = *Commit*(*msg*, *open*), where

  msg is some yet-secret message,

  and *open* is yet-secret random value.

- Later on, the Originator makes (*msg*, *open*) public.
  - Verifiers can then check that indeed:

    C = *Commit*(*msg*, *open*).

Security Requirements

- **Binding**
  - Originator cannot change *msg* after having announced *C*.
- **Hiding**
  - Without a help of the originator, recipient of *C* cannot obtain any non-negligible information on *msg*.
- We cannot have both *perfect* binding and *perfect* concealing.
  - We can, however, achieve a secure enough settlement.

Keyed Bit Commitment

- Defines $Commit_{\boldsymbol{Key}}(msg, open)$
  - Only the parties knowing the ***Key*** can participate in the protocol.
  - Others can only gain or introduce a (pseudo)random noise from/to the protocol flow.

WPS-Style Scheme

$$Commit_{Key}(msg, open) =$$

$$= \text{HMAC-SHA-256}_{Key}(open \,||\, msg),$$

where $open \in_R \{0, 1\}^{128}$.

Part THREE
The Registration Protocol

Device Password Verification

- Establishes mutual authentication for interfaces *E* and *M*.
  - Both parties are initially untrusted.
  - There is also an implicit key agreement securing the ongoing management messages.
  - PIN is a special kind of decimal-only DP.
- Defines two entities: **Registrar and Enrollee**.
  - Directly fits interface *E*.
  - For interface *M*, the Enrollee's role is played by AP.

Registration Protocol Phases

1. Ephemeral Diffie-Hellman key agreement
   - Steps M1, M2
   - Provides envelope keys: *AuthKey*, *KeyWrapKey*, and *EMSK* (Extended Master Session Key).
2. **Device Password mutual verification**
   - Steps M3, …, M7-A
   - Uses keyed Bit Commitment variant.
   - We focus solely on this part.
3. Configuration data exchange
   - Steps M7-B, M8
   - There can also be persistent master session that continually governs the AP (using *EMSK*).

Step M3

Enrollee **Registrar**

PKInf … Diffie-Hellman public key info (ephemeral)
DP_L … left half of Device Password, $|DP_L| = [|DP| + (|DP| \bmod 2)]/2$
DP_R … right half of Device Password, $|DP_R| = |DP| - |DP_L|$

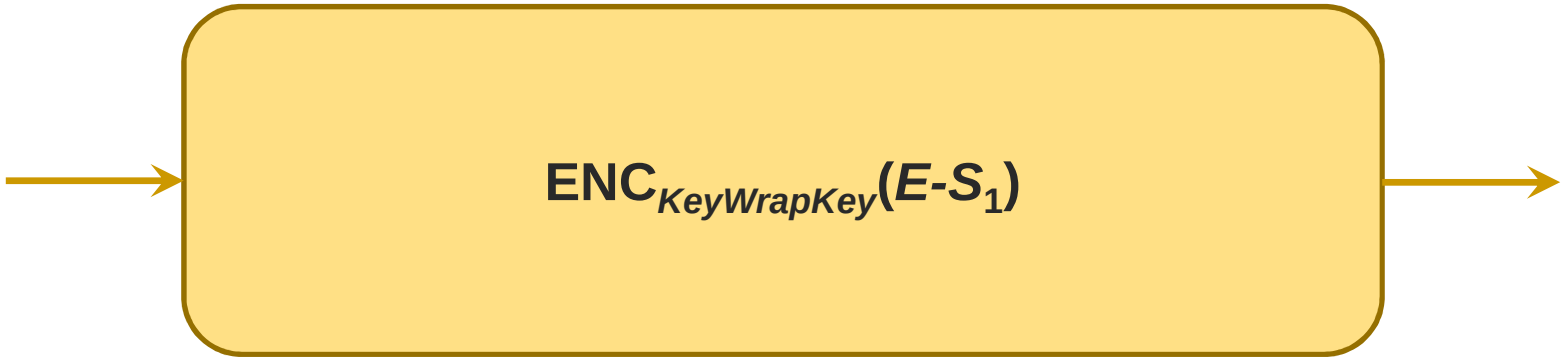
Step M4

Enrollee **Registrar**

PKInf … Diffie-Hellman public key info (ephemeral)
DP_L … left half of Device Password
DP_R … right half of Device Password, $|DP_R| \leq |DP_L|$

Step M5

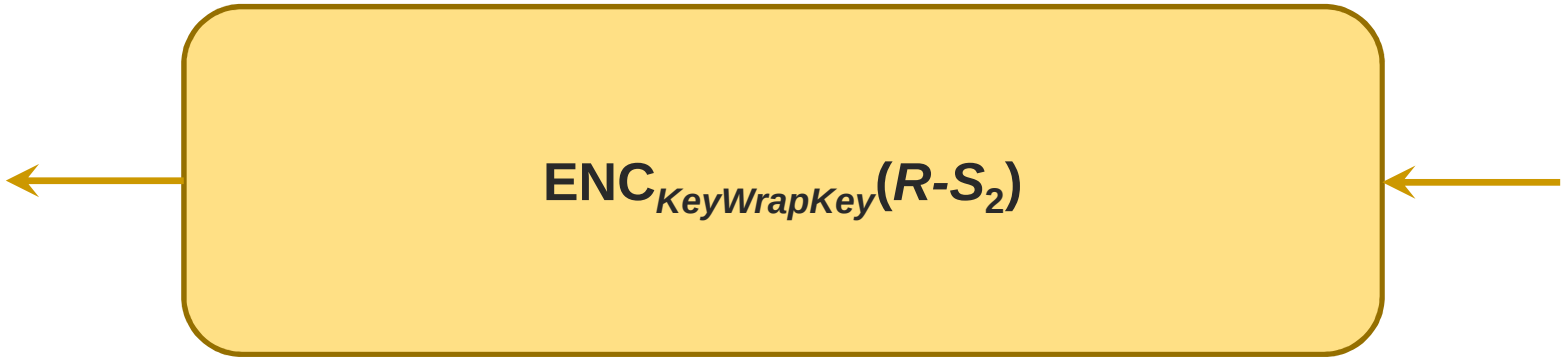
Enrollee **Registrar**



Finishes the mutual authentication (confirmation) of the first half of Device Password.

Step M6

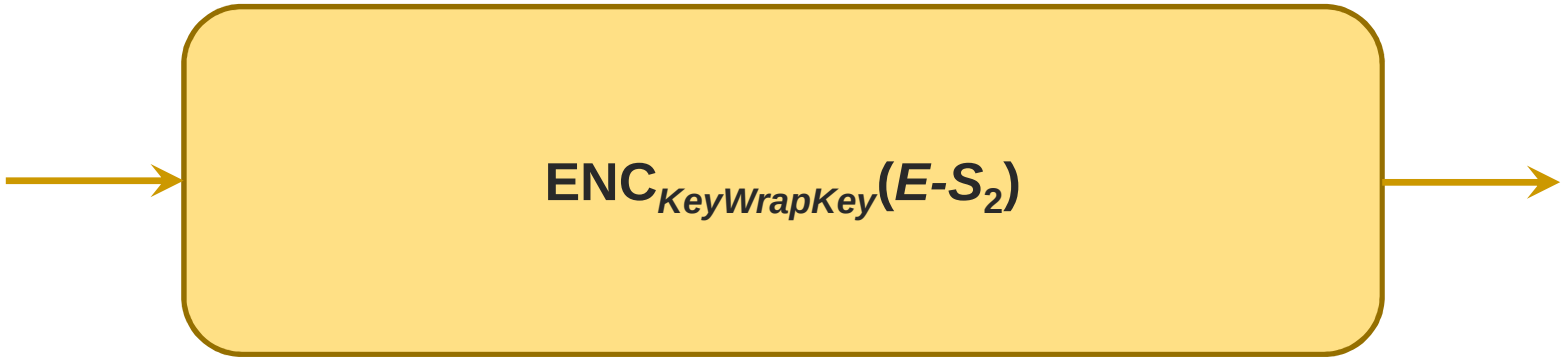
Enrollee **Registrar**



Starts the mutual confirmation of the second half of Device Password.

Step M7

Enrollee **Registrar**



Finishes the mutual confirmation of the second half of Device Password.

Continual Verification

- After each message exchange, the recipient gradually performs cryptographic checks of available data to decide whether to proceed in the protocol flow or stop. [7]

Attack In M4, M6

- Active *online* brute force on PIN by fraudulent Registrar.
  - Independently by Stefan Viehboeck [5] and Tactical Network Solutions in 2011 [9].
  - In 2008(!), described by Lindell for SSP in BT.[3]
- Online attacker repeatedly queries Enrollee (usually AP), until PIN is found.
  - Splitting Device Password induces $O(10^{N/2})$.
  - **Requires intensive protocol restarting.**
  - Already anticipated by WFA in 2006! [6], [7]

Dual Attack In M4, M6

- Active *offline* brute force on PIN by fraudulent Enrollee.
  - Not so broadly exploited, yet.
  - Neither for Bluetooth.
- Attacker receives R-S_1 (R-S_2) and searches *offline* for valid PIN.
  - $O(10^{N/2})$, not limited (or even limitable) by peer speed.
  - **Requires two protocol restarts** plus one for utilizing the PIN.
  - Already anticipated by WFA! [6], [7]

Static PIN Assumption

- Both attacks require the Registration Protocol to be restarted.
  - Thanks(!) to the split verification.
- **Generating fresh PIN defeats it all.**
  - However, (quasi) static PIN is sometimes necessary – especially in IBSS. [2]
  - At least, we shall follow technical procedures of the standard then. [2], [7]

Cousin Bluetooth “Classic”

- Passkey-based Secure Simple Pairing employs the same idea [1], but:
  - Despite having D-H ready in place, the Bit Commitment is not D-H-keyed!
    - We can brute-force the PIN basing on *passive* interception. [3]
  - The verification is “over split” to bit-by-bit!
    - Online/offline *k*-bit PIN brute-force in just *O*(*k*).
  - So, the risk of *static* PIN compromise in SSP is much higher than for WPS.
    - It is even higher than for Bluetooth 2.0!
    - Despite this, Secure Simple Pairing is quite popular…

Bluetooth Low Energy

- Even worse re-incarnation of the same idea.
  - Bit Commitment is not keyed.
    - There is even no D-H at all, so not only the PIN, but the whole link key is at risk [11].
  - Passkey is verified at once (no split).
    - Interestingly, the two Bluetooth standards exercise the two limit splitting strategies. Both wrong for *static* PIN.
  - **BLE is obviously designed to be very simple and relies on many assumptions.**
    - Fresh PIN anytime pairing is started.
    - No passive adversary during pairing.

Bit Commitment á-la BLE

- Defines $msg = (TK, p_1, p_2)$, where
  - TK is a reformatted PIN value,
  - p_1 and p_2 are certain known labels.
  - Furthermore $open = rand$.
  - Then $\boldsymbol{C = Commit(msg, open) =}$

$$\boldsymbol{= AES_{TK}[AES_{TK}(open \oplus p_1) \oplus p_2]}$$

Lack of Binding in BLE BC

- Having been given any (modified) *msg* and a former commitment *C*,
  - we can trivially find its new corresponding opener *rand* as

    $$\boldsymbol{rand = AES^{-1}_{TK}[AES^{-1}_{TK}(C) \oplus p_2] \oplus p_1}$$

  - Actually, this is just a straight-forward CBC-MAC inversion towards *IV* for a known key and message.

One-Time PIN is Broken

- Besides those weaknesses already noted in BT Core Spec. [1] and elaborated in [11], this is a new attack.
  - Even purely One-Time PIN authentication can be broken in BLE.
  - Due to the lack of binding, the attacker can utilize a cracked PIN just in the very same pairing procedure.

BLE_decommit(...) PoC

```
Demonstrating the crypto exploit ------------
!new TK!     : 0000000000000000000000001E240
p1           : 05000800000302070710000001010001
p2           : 00000000A1A2A3A4A5A6B1B2B3B4B5B6
commitment   : 1e1e3fef878988ead2a74dc5bef13b86
!new rand!   : e27fced1adb1f637c3bbb4eaa0ba409a

recomputed commitment :
                    1e1e3fef878988ead2a74dc5bef13b86
X-check: OK, commitment is still(!) valid
```

Python source at: http://crypto.hyperlink.cz/files/blecommit.py

Countermeasure

- Assuming only *open* can change after sending the commitment *C*, there are several ways to a trivial fix.

Either

$$C = AES_{TK \oplus open}[AES_{TK \oplus open}(open \oplus p_1) \oplus p_2]$$

Or

$$C = AES_{TK}[AES_{TK}(open \oplus p_1) \oplus open \oplus p_2]$$

Or Even

$$C = AES_{TK}[AES_{TK}(open \oplus p_1) \oplus p_2] \oplus open$$

Apropos Bluetooth…

Affordable Bluetooth hacking tool is here!


Part FOUR
The Swamp

Ideal Swamp

- Recognizing that you are in the swamp is unavoidable…
  - …but not sufficient to get out of there.
  - You either know the way out or not.
- There is no guiding information you can rely on.
  - **The swamp does not want to help you.**

WPS Patching Obstacles

- Or, why this effort usually fails?
  - Uneasy to correctly stop WPS flow without leaking *any* useful information.
  - PIN-masking is of no help:
    - The mask function – whatever it is – must be still computable by any active WLAN entity, so it is no obstacle for active brute force.
    - Joining both PIN halves facilitates the dual attack.
  - Furthermore, it is impractical to patch all devices. Usually, we can patch AP only.

Swamp

- WPS *PIN*-verification hardening in case of standard countermeasures do not help.
  - E.g. autonomous, loosely supervised AP with *static* PIN and high availability demands.
- Addresses the original attack [5] by patching the Enrollee (AP) only.
  - **No need to patch the Registrars.**
  - **Reinstalls $O(10^N)$ complexity.**
  - **Preserves reasonable resistance against the dual attack.**

New Registration Protocol

- Two phases at the Enrollee (AP) side.
  - **Brave Step.** We behave according to the original standard, so giving the Registrar a chance to authenticate.
  - **Swamp Walk.** After a few failed Brave Steps, we start behaving according to the Swamp rules.
    - *Swamp Walk terminates either by a successful authentication, long timeout, or supervisor intervention.*

Brave Step

- Necessary to:
  - handle ephemeral PIN,
  - prevent honest users annoyance.
- Only just a few steps shall be allowed.
  - Say three attempts before swamp.
  - **The number is settlement between comfort and information leakage.**

Swamp Walk – Init

- Set $\boldsymbol{DP_L^* = NaN}$

  NaN ~ anything that is Not a Number

Swamp Walk – Step M3

- If DP_L^* == NaN
  - Send random $E\text{-}Hash_1$ and $E\text{-}Hash_2$.
- Else
  - Send correct commitments $E\text{-}Hash_1$ and $E\text{-}Hash_2$ for DP_L^* and DP_R, respectively.
  - Note DP_L **(!star!)** instead of DP_L.

Swamp Walk – Step M4

- **Performed by the Registrar with no change.**

Swamp Walk – Step M5

- If DP_L^* == NaN
  - Derive (offline brute-force) DP_L^* from $R\text{-}Hash_1$ and $R\text{-}S_1$.
  - If impossible, let DP_L^* = NaN.
  - Send “Failed” (WSC_NACK).

First attempt always fails, regardless of DP_L used by Registrar. So, the attacker can recognize the swamp, but it is of no help. They need to follow the rules. Furthermore, it does not disturb honest Registrar too much – it just uses the quasi-static PIN once more…

Swamp Walk – Step M5

- If DP_L^* != NaN
  - Verify (standard way) that DP_L^* conforms with $R\text{-}Hash_1$ and $R\text{-}S_1$.
  - If verified positively
    - Send $\mathbf{ENC_{KeyWrapKey}(E\text{-}S_1)}$ and continue the flow.
  - Else
    - Derive (offline brute-force) DP_L^* from $R\text{-}Hash_1$ and $R\text{-}S_1$.
    - If impossible, let DP_L^* = NaN.
    - Send “Failed” (WSC_NACK).

Swamp Walk – Step M6

- **Performed by the Registrar with no change.**

Swamp Walk – Step M7

- If $DP_L^* == DP_L$
  - Verify (standard way) that DP_R conforms with $R\text{-}Hash_2$ and $R\text{-}S_2$.
  - If verified positively
    - Send $\mathbf{ENC_{KeyWrapKey}(E\text{-}S_2)}$ and continue the flow.
  - Else
    - Send “Failed” (WSC_NACK).

Swamp Walk – Step M7

- If DP_L^* != DP_L
  - *Perform some dummy computation.*
    - This is to prevent timing attacks.
  - Send “Failed” (WSC_NACK).

Swamp Walk Remarks

- We shall still pay attention to brute-force feasibility.
  - The PIN entropy still matters.
  - We still need to enforce a safe response rate.
  - We shall monitor persistent active attackers.
- Swamp is just a significant sidekick, not a whole solution.
  - It helps by reinstalling the “full exponential” brute force complexity.
  - In other words, it reliefs the pain introduced by the PIN splitting which was, however, necessary to defeat the dual attack…

How About Cousin Bluetooth?

- Secure Simple Pairing can be patched in the same way.
  - Helps to cope with the active online adversary playing the Initiator role.
  - Reinstalls $O(2^k)$ complexity.
- Needs just a slight tailoring for the higher splitting rate.
  - **Blue Swamp…**
  - Hint: Manage all but the final PIN bits in the same way as DP_L^*.

Conclusion

- Wi-Fi Protected Setup employs reasonable cryptographic protocol for practically feasible and yet-secure PIN-based key agreement.
  - The key can be then used to manage network credentials.
  - Same principle as Secure Simple Pairing in Bluetooth. With one exception – WPS is more secure.
- Special care must be taken for *static* PIN.
  - We shall read the standard – it is almost all in there!
  - **Anyway, this is a vital place where to look for penetration tests inspiration.**

Acknowledgement

Special thanks goes to Tomáš Jabůrek and Radek Komanický of Raiffeisenbank CZ for continual support.

Thank You For Attention

Tomáš Rosa, Ph.D.
http://crypto.hyperlink.cz

References

1. *Bluetooth Core Specification*, ver. 4.0, Bluetooth SIG, June 2010
2. *IBSS with Wi-Fi Protected Setup*, Technical Specification, ver. 1.0.0, Wi-Fi Alliance, November 2012
3. Lindell, A.-Y.: *Attacks on the Pairing Protocol of Bluetooth v2.1*, Black Hat USA 2008, June 2008
4. Lulev, H.: *Overview of Bit Commitment Schemes*, Bachelor Thesis at Darmstadt University of Technology, Department of Computer Science, December 2007
5. Viehboeck, S.: *Brute forcing Wi-Fi Protected Setup – When Poor Design Meets Poor Implementation*, ver. 3, December 2011
6. *Wi-Fi Protected Setup Specification*, ver. 1.0h, Wi-Fi Alliance, December 2006
7. *Wi-Fi Simple Configuration*, Technical Specification, ver. 2.0.2, Wi-Fi Alliance, January 2012
8. http://www.wi-fi.org/knowledge-center/articles/wi-fi-protected-setup [retrieved May-02-2013]
9. http://www.tacnetsol.com/news/2011/12/28/cracking-wifi-protected-setup-with-reaver.html, December 2011 [retrieved May-02-2013]
10. http://ubertooth.sourceforge.net/ [retrieved May-02-2013]
11. Ryan, M.: *How Smart is Bluetooth Smart?*, Shmoocon 2013, Feb 16th, http://lacklustre.net/ [retrieved May-16-2013]