- Agenda:
  1. a 2-part unit summary:
     - recap re. motivation, i.e., why the unit exists,
     - what did and didn't we do in the unit,
  2. drop-in slot re. coursework assignment.

A real-world story: an attack [2] on TLS 1.2 + OpenSSL 0.9.8*g* (1)

Quote

> *The function* `BN_nist_mod_384` *(in* `crypto/bn/bn_nist.c`*) gives wrong results for some inputs.*
>
> – *Reimann [5]*

Algorithm (NIST-P-256-Reduce, per Solinas [6, Example 3, Page 20])

Input: For $w = 32$-bit words, a 16-word integer product $z = x \cdot y$ and the modulus $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$
Output: The result $r = z \pmod{p}$

1. Form the nine, 8-word intermediate variables

$$
\begin{array}{lclcccccccccc}
S_0 & = & \langle & z_0, & z_1, & z_2, & z_3, & z_4, & z_5, & z_6, & z_7 & \rangle \\
S_1 & = & \langle & 0, & 0, & 0, & z_{11}, & z_{12}, & z_{13}, & z_{14}, & z_{15} & \rangle \\
S_2 & = & \langle & 0, & 0, & 0, & z_{12}, & z_{13}, & z_{14}, & z_{15}, & 0 & \rangle \\
S_3 & = & \langle & z_8, & z_9, & z_{10}, & 0, & 0, & 0, & z_{14}, & z_{15} & \rangle \\
S_4 & = & \langle & z_9, & z_{10}, & z_{11}, & z_{13}, & z_{14}, & z_{15}, & z_{13}, & z_8 & \rangle \\
S_5 & = & \langle & z_{11}, & z_{12}, & z_{13}, & 0, & 0, & 0, & z_8, & z_{10} & \rangle \\
S_6 & = & \langle & z_{12}, & z_{13}, & z_{14}, & z_{15}, & 0, & 0, & z_9, & z_{11} & \rangle \\
S_7 & = & \langle & z_{13}, & z_{14}, & z_{15}, & z_8, & z_9, & z_{10}, & 0, & z_{12} & \rangle \\
S_8 & = & \langle & z_{14}, & z_{15}, & 0, & z_9, & z_{10}, & z_{11}, & 0, & z_{13} & \rangle
\end{array}
$$

2. Compute

$$r = S_0 + 2S_1 + 2S_2 + S_3 + S_4 - S_5 - S_6 - S_7 - S_8 \pmod{p}.$$

3. Return $0 \leq r < p$.

Algorithm (NIST-P-256-Reduce, per OpenSSL 0.9.8g)

Input: For $w = 32$-bit words, a 16-word integer product $z = x \cdot y$ and the modulus $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$
Output: The (potentially incorrect) result $r = z \pmod{p}$

1. Form the nine, 8-word intermediate variables

$$
\begin{array}{lcllllllllll}
S_0 & = & \langle & z_0, & z_1, & z_2, & z_3, & z_4, & z_5, & z_6, & z_7 & \rangle \\
S_1 & = & \langle & 0, & 0, & 0, & z_{11}, & z_{12}, & z_{13}, & z_{14}, & z_{15} & \rangle \\
S_2 & = & \langle & 0, & 0, & 0, & z_{12}, & z_{13}, & z_{14}, & z_{15}, & 0 & \rangle \\
S_3 & = & \langle & z_8, & z_9, & z_{10}, & 0, & 0, & 0, & z_{14}, & z_{15} & \rangle \\
S_4 & = & \langle & z_9, & z_{10}, & z_{11}, & z_{13}, & z_{14}, & z_{15}, & z_{13}, & z_8 & \rangle \\
S_5 & = & \langle & z_{11}, & z_{12}, & z_{13}, & 0, & 0, & 0, & z_8, & z_{10} & \rangle \\
S_6 & = & \langle & z_{12}, & z_{13}, & z_{14}, & z_{15}, & 0, & 0, & z_9, & z_{11} & \rangle \\
S_7 & = & \langle & z_{13}, & z_{14}, & z_{15}, & z_8, & z_9, & z_{10}, & 0, & z_{12} & \rangle \\
S_8 & = & \langle & z_{14}, & z_{15}, & 0, & z_9, & z_{10}, & z_{11}, & 0, & z_{13} & \rangle
\end{array}
$$

2. Compute

$$
\begin{array}{lcl}
S & = & S_0 + 2S_1 + 2S_2 + S_3 + S_4 - S_5 - S_6 - S_7 - S_8 \\
  & = & t + c \cdot 2^{256}
\end{array}
$$

3. Compute

$$
\begin{array}{lcll}
r & = & t - c \cdot p & \pmod{2^{256}} \\
  & = & t - \mathrm{sign}(c) \cdot T[|c|] & \pmod{2^{256}}
\end{array}
$$

for pre-computed $T[i] = i \cdot p$.

4. If $r \geq p$ (resp. $r < 0$) then update $r \leftarrow r - p$ (resp. $r \leftarrow r + p$), return r.

- Observation(s):
  - good: `BN_nist_mod_256` (resp. `BN_nist_mod_384`) is more efficient.
  - bad: `BN_nist_mod_256` (resp. `BN_nist_mod_384`) can produce an incorrect result, e.g.,
    1. triggered deliberately with special-form operands

$$
\begin{array}{rcl}
x & = & (2^{32} - 1) \cdot 2^{224} + 3 \cdot 2^{128} + x_0 \\
y & = & (2^{32} - 1) \cdot 2^{224} + 1 \cdot 2^{96} \;\; + y_0
\end{array}
$$

       for random $0 \leq x_0, y_0 < 2^{32}$, *or*
    2. triggered randomly with probability $\sim 10 \cdot 2^{-29}$.

A real-world story: an attack [2] on TLS 1.2 + OpenSSL 0.9.8g (4)

Issue 2: (opt-out) ephemeral-static EC-DHE

Algorithm (EC-DH(E) key agreement [7, Section 8.1][8, Section 2.1])

$\mathcal{A}$		$\mathcal{B}$
Knows $\mathbb{G} = E(\mathbb{F}_q) = \langle G \rangle$ of order n, $pk_{\mathcal{B}}$, $(pk_{\mathcal{A}})^{\dagger}$, $(sk_{\mathcal{A}})^{\dagger}$		Knows $\mathbb{G} = E(\mathbb{F}_q) = \langle G \rangle$ of order n, $(pk_{\mathcal{A}})^{\dagger}$, $pk_{\mathcal{B}}$, $sk_{\mathcal{B}}$
$k_{\mathcal{A}}^{(i)} \xleftarrow{\$} \{1, 2, \ldots, n-1\}$ $Q_{\mathcal{A}}^{(i)} \leftarrow \left[k_{\mathcal{A}}^{(i)}\right] G$		$k_{\mathcal{B}}^{(i)} \xleftarrow{\$} \{1, 2, \ldots, n-1\}$ $Q_{\mathcal{B}}^{(i)} \leftarrow \left[k_{\mathcal{B}}^{(i)}\right] G$
	$\xrightarrow{Q_{\mathcal{A}}^{(i)}}$	
	$\xleftarrow{Q_{\mathcal{B}}^{(i)}}$	
$R_{\mathcal{A}}^{(i)} \leftarrow \left[k_{\mathcal{A}}^{(i)}\right] Q_{\mathcal{B}}^{(i)} = \left[k_{\mathcal{A}}^{(i)} \cdot k_{\mathcal{B}}^{(i)}\right] G$		$R_{\mathcal{B}}^{(i)} \leftarrow \left[k_{\mathcal{B}}^{(i)}\right] Q_{\mathcal{A}}^{(i)} = \left[k_{\mathcal{B}}^{(i)} \cdot k_{\mathcal{A}}^{(i)}\right] G$
Use $R_{\mathcal{A}}^{(i)}$		Use $R_{\mathcal{B}}^{(i)}$

A real-world story: an attack [2] on TLS 1.2 + OpenSSL 0.9.8g (4)

Issue 2: (opt-out) ephemeral-static EC-DHE

Algorithm (EC-DH(E) key agreement [7, Section 8.1][8, Section 2.4])

$\mathcal{A}$		$\mathcal{B}$
Knows $\mathbb{G} = E(\mathbb{F}_q) = \langle G \rangle$ of order n, $pk_{\mathcal{B}}$, $(pk_{\mathcal{A}})^{\dagger}$, $(sk_{\mathcal{A}})^{\dagger}$		Knows $\mathbb{G} = E(\mathbb{F}_q) = \langle G \rangle$ of order n, $(pk_{\mathcal{A}})^{\dagger}$, $pk_{\mathcal{B}}$, $sk_{\mathcal{B}}$
$k_{\mathcal{A}} \xleftarrow{\$} \{1, 2, \ldots, n-1\}$ $Q_{\mathcal{A}} \leftarrow [k_{\mathcal{A}}]\, G$		$k_{\mathcal{B}} \xleftarrow{\$} \{1, 2, \ldots, n-1\}$ $Q_{\mathcal{B}} \leftarrow [k_{\mathcal{B}}]\, G$
	$\xrightarrow{Q_{\mathcal{A}}}$	
	$\xleftarrow{Q_{\mathcal{B}}}$	
$R_{\mathcal{A}}^{(i)} \leftarrow [k_{\mathcal{A}}]\, Q_{\mathcal{B}} = [k_{\mathcal{A}} \cdot k_{\mathcal{B}}]\, G$		$R_{\mathcal{B}}^{(i)} \leftarrow [k_{\mathcal{B}}]\, Q_{\mathcal{A}} = [k_{\mathcal{B}} \cdot k_{\mathcal{A}}]\, G$
Use $R_{\mathcal{A}}^{(i)}$		Use $R_{\mathcal{B}}^{(i)}$

Algorithm (EC-DH(E) key agreement [7, Section 8.1][8, Section 2.3])

$\mathcal{A}$		$\mathcal{B}$
Knows $\mathbb{G} = E(\mathbb{F}_q) = \langle G \rangle$ of order n, $pk_{\mathcal{B}}$, $(pk_{\mathcal{A}})^{\dagger}$, $(sk_{\mathcal{A}})^{\dagger}$		Knows $\mathbb{G} = E(\mathbb{F}_q) = \langle G \rangle$ of order n, $(pk_{\mathcal{A}})^{\dagger}$, $pk_{\mathcal{B}}$, $sk_{\mathcal{B}}$
		$k_{\mathcal{B}} \xleftarrow{\$} \{1, 2, \ldots, n-1\}$ $Q_{\mathcal{B}} \leftarrow [k_{\mathcal{B}}]\, G$
$k_{\mathcal{A}}^{(i)} \xleftarrow{\$} \{1, 2, \ldots, n-1\}$ $Q_{\mathcal{A}}^{(i)} \leftarrow \left[k_{\mathcal{A}}^{(i)}\right] G$		
	$\xrightarrow{Q_{\mathcal{A}}^{(i)}}$	
	$\xleftarrow{Q_{\mathcal{B}}}$	
$R_{\mathcal{A}}^{(i)} \leftarrow \left[k_{\mathcal{A}}^{(i)}\right] Q_{\mathcal{B}} = \left[k_{\mathcal{A}}^{(i)} \cdot k_{\mathcal{B}}\right] G$		$R_{\mathcal{B}}^{(i)} \leftarrow [k_{\mathcal{B}}]\, Q_{\mathcal{A}}^{(i)} = \left[k_{\mathcal{B}} \cdot k_{\mathcal{A}}^{(i)}\right] G$
Use $R_{\mathcal{A}}^{(i)}$		Use $R_{\mathcal{B}}^{(i)}$

A real-world story: an attack [2] on TLS 1.2 + OpenSSL 0.9.8g (5)
Issue 2: (opt-out) ephemeral-static EC-DHE

- Observation(s):
  - good: the key agreement is more efficient (for the server).
  - good: input points are validated by testing whether

$$P_y^2 \stackrel{?}{=} P_x^3 + a_4 P_x + a_6$$

    given $P = (P_x, P_y)$.
  - bad: ephemeral-*static* EC-DHE is the default i.e.,
    - uses a per-invocation (of the library) rather than a per-session key, *unless*
    - one explicitly uses `SSL_CTX_set_options` using `SSL_OP_SINGLE_ECDH_USE`

    which means $k_{\mathcal{B}}$ is a static, fixed target for any attack.
  - bad: if we select $P = (P_x, P_y)$ as follows
    1. Select P_x such that during the computation of the RHS $t' = (P_x^2 + a_4) \cdot P_x + a_6 \pmod{p}$
       - the step $t'_0 = P_x^2 \pmod{p}$ *does not* trigger the bug, and
       - the step $t'_1 = (t'_0 + a_4) \cdot P_x \pmod{p}$ *does* trigger the bug, and
       - t' is a quadratic residue modulo p.
    2. Compute $P_y = \sqrt{t'} \pmod{p}$.

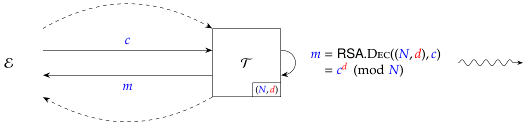
    then P passes validation, but is on some curve E' rather than E.

Quote

Decrypting ciphertexts on any computer which multiplies even one pair of numbers incorrectly can lead to full leakage of the secret key, sometimes with a single well-chosen ciphertext.

– Biham et. al. [1, Page 1]

- Scenario:
  - given the following interaction between an **attacker** $\mathcal{E}$ and a **target** $\mathcal{T}$




```
algorithm RSA.Dec((N, d), c) begin
  Pre-compute Π = (N, ρ, ω) from N
  ĉ ← MontMul(Π, c, ρ^2 mod N)
  r̂ ← MontMul(Π, c, ρ^2 mod N)
  for i = |d| − 2 downto 0 do
    r̂ ← MontMul(Π, r̂, r̂)
    if d_i = 1 then
      r̂ ← MontMul(Π, r̂, ĉ)
    end
  end
  return MontMul(Π, r̂, 1)
end
```

  - and noting that
    - there are no countermeasures implemented,
    - the Montgomery multiplication implementation is FIOS-based [3],
    - the $(w \times w)$-bit integer multiplier hardware has a bug: when computing $r = x \times y$ if

$$\begin{array}{lcccl} x \neq \alpha & \vee & y \neq \beta & \Rightarrow & r \text{ is correct} \\ x = \alpha & \wedge & y = \beta & \Rightarrow & r \text{ is } \textit{incorrect} \end{array}$$

      for some known (but arbitrary) α and β.
  - how can $\mathcal{E}$ mount a successful attack, i.e., recover d ?

- Attack [1, Section 4.2]:
  - in some t-th step, $\mathcal{E}$
    - knows some more-significant portion of the binary expansion of d, and
    - aims to recover d_t, the next less-significant unknown bit,
  - select a c so during decryption when $i = t$ and just after line #6

$$\begin{array}{lll} \exists\, j & \text{such that} & \hat{r}_j = \alpha \\ \exists\, j & \text{such that} & \hat{c}_j = \beta \end{array}$$

    i.e., α and β occur in the representations of $\hat{r}$ and $\hat{c}$,
  - this selection means

$$\begin{array}{lllll} d_t = 0 & \Rightarrow & \hat{r} \text{ is not multiplied by } \hat{c} & \Rightarrow & \text{the bug is not triggered} \\ d_t = 1 & \Rightarrow & \hat{r} \text{ is multiplied by } \hat{c} & \Rightarrow & \text{the bug is triggered} \end{array}$$

  - test whether

$$m^e \pmod{N} \stackrel{?}{=} c$$

    and infer

$$\begin{array}{lllll} m \text{ is correct} & \Rightarrow & \text{the bug was not triggered} & \Rightarrow & d_t = 0 \\ m \text{ is incorrect} & \Rightarrow & \text{the bug was triggered} & \Rightarrow & d_t = 1 \end{array}$$

Feature	Biham et. al. [1, Section 4.2]	Brumley et. al. [2, Section 3]
Target	Fixed d	Fixed $k_{\mathcal{T}}$
Input	Arbitrary poisoned integer $c \in \mathbb{Z}_N^*$	Controlled distinguisher point $Q_{\mathcal{E}} = [k_{\mathcal{E}}]\,G \in E(\mathbb{F}_p)$
Computation	Left-to-right binary exponentiation	Left-to-right (modified) wNAF scalar multiplication
Leakage	Re-encrypt m using e, check against c	Handshake success/failure

- Epilogue:
  - good(ish):

Quote

We appreciate you reporting this issue to us but, unfortunately, we aren't inclined to handle this vulnerability because it is already patched and only affects obsolete Linux distributions.

– CERT

https://jscholarship.library.jhu.edu/items/00b58834-a88c-449e-ab23-db2f44207383

- Epilogue:

  - bad: even circa 2013, the reality [4] seemed to differ somewhat:

Version	Percentage
0.9.8e-fips-rhel5	37.25
0.9.8g	14.50
0.9.7a	7.02
0.9.8o	4.76
1.0.0-fips	4.36
0.9.7d	2.91
0.9.8n	2.75
0.9.7e	1.94
0.9.8c	1.80
0.9.8m	1.74
0.9.8e	1.72
0.9.8r	1.71

Table 2: Most popular OpenSSL versions on the Internet.

Distribution	OSSL Version	CVEs
Debian Squeeze (6.0)	0.9.8o	11
Debian Lenny (5.0)	0.9.8g	24
Debian Etch (4.0)	0.9.8c	26
RHEL 6	0.9.8e/1.0.0-fips	0/14
RHEL 5	0.9.7a/0.9.8e-fips	14/0
RHEL 4	0.9.6b/0.9.7a	9/14
Fedora 18	1.0.1c	3
Fedora 17	1.0.0i	3
Fedora 16	1.0.0e	9

Table 3: Default OpenSSL versions shipping with popular Linux distributions.

https://jscholarship.library.jhu.edu/items/00b58834-a88c-449e-ab23-db2f44207383

Unit summary (1)

- Summary:

https://memegenerator.net

Unit summary (2)

- Summary: what *have* we done includes

1. focused on some high-level outcomes:
    - improved

$$\left.\begin{array}{c}\text{awareness}\\\text{understanding}\\\text{skills}\\\vdots\end{array}\right\} \Rightarrow \text{ability to engage with problems, produce solutions, ...}$$

    - general concepts (versus specific examples) ⇒ long-term (versus short-term) value.

2. highlighted some high-level principles:
    - most effective implementation will be domain-specific,
    - apply adversarial thinking to *everything*,
    - need for and value in well-considered trade-offs,
    - don't *over*-optimise to the point efficiency > security,
    - apply "*inverse* Postel's Law", i.e., be *very* strict re. what you accept as input,
    - ...

3. exposed some low-level detail:
    - tools, techniques, and technologies,
    - shift from abstract toward and including concrete (e.g., AES versus generic block cipher),
    - written standards, RFCs, etc. (e.g., FIPS-197 versus lecture slides),
    - ...

Unit summary (3)

- Summary: what *haven't* we done includes

1. greater *depth*, i.e., more X for $X \in$ COMS30048:
    - more implementation
        - platforms (e.g., FPGAs, ASICs, GPUs, ..., JavaScript versus C)
        - constraints (e.g., from use-case, platform, tooling, ...)
        - co-design (e.g., hardware/software, specification/implementation, ...)
        - ...
    - more attacks
    - more countermeasures
    - more primitives (e.g., PQC, LWC, hash functions, ..., FHE, MPC, ...)
    - more protocols (e.g., DNSSEC, IPSec, ...)
2. greater *breadth*, i.e., more X for $X \notin$ COMS30048:
    - hardware security (e.g., TEEs, HSMs, secure boot and update, FDE, ...)
    - formal verification
    - key management (e.g., secure generation, storage, and erasure, ...)
    - social-technical (e.g., usability, politics, risk analysis, supply chain, disclosure, ...)
    - certification and standardisation processes
    - ...

References

[1] E. Biham, Y. Carmeli, and A. Shamir. "Bug Attacks". In: *Advances in Cryptology (CRYPTO)*. Vol. 5157. LNCS. Springer-Verlag, 2008, pp. 221–240 (see pp. 10, 12, 13).

[2] B. Brumley et al. "Practical realisation and elimination of an ECC-related software bug attack". In: *Topics in Cryptology (CT-RSA)*. LNCS 7178. Springer-Verlag, 2012, pp. 171–186 (see pp. 2–15).

[3] Ç.K. Koç, T. Acar, and B.S. Kaliski. "Analyzing and comparing Montgomery multiplication algorithms". In: *IEEE Micro* 16.3 (1996), pp. 26–33 (see p. 11).

[4] P.D. Martin et al. *Classifying Network Protocol Implementation Versions: An OpenSSL Case Study*. Tech. rep. 13-01. Johns Hopkins University, 2013. URL: http://www.michaelrushanan.org/pdf/martin.pdf (see pp. 14, 15).

[5] H. Reimann. `BN_nist_mod_384` *gives wrong answers*. `openssl-dev` mailing list #1593. 2007. URL: http://marc.info/?t=119271238800004 (see p. 2).

[6] J.A. Solinas. *Generalized Mersenne Numbers*. Tech. rep. CORR 99-39. Centre for Applied Cryptographic Research (CACR), University of Waterloo, 1999 (see p. 3).

[7] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol version* 1.2. Internet Engineering Task Force (IETF) Request for Comments (RFC) 5246. 2008. URL: http://tools.ietf.org/html/rfc5246 (see pp. 6–8).

[8] E. Rescorla. *Diffie-Hellman Key Agreement Method*. Internet Engineering Task Force (IETF) Request for Comments (RFC) 2631. 1999. URL: http://tools.ietf.org/html/rfc2631 (see pp. 6–8).