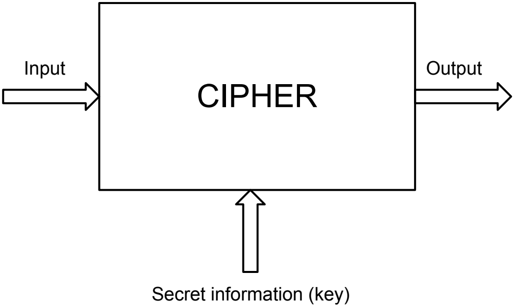
# AES Timing Attacks

Hardware and Software Design for Cryptographic Applications

April 11, 2013

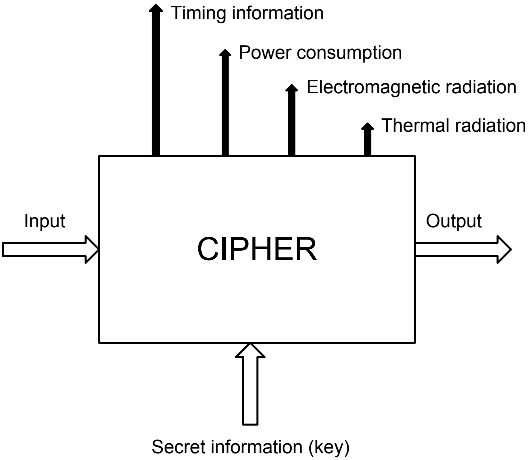
# Ciphers as a Black Box

In theory, encryption (and decryption) implementations operate as black boxes.



# Information Leakage

In reality, it's hard to prevent additional information from being leaked at runtime.



# Side Channel Attacks

**Definition**: Any attack on a cryptosystem using information leaked given off as a byproduct of the physical implementation of the cryptosystem, rather than a theoretical weakness [1], is a *side channel attack*.

We focus on **timing attacks** for **software implementations** of AES.

We assume the attacker can *easily* capture this timing information.

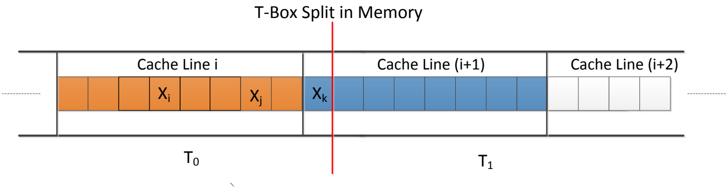
# History of Timing Attacks on Cryptographic Primitives

- LIST OF PAPERS FROM BONNEAU HISTORY

# Timing Attacks on AES

- Rijndael was deemed not susceptible to timing attacks in the AES contest
- AES targeted attacks can be based on *statistical* evidence [2].
  - Observation: The entire encryption time can be affected by the input bytes $p_i^0 \oplus k_i^0$
    - Why?
  - Step 1: Capture timing data on a reference and target machine for each value of a particular input byte $p_i^0 \oplus k_i^0 = x_i^0$
  - Step 2: Perform correlation between reference and target data
  - Step 3: Dance.
- Or they can be more targeted:
  - Exploit relationships between secret information of the primitive and known data.
  - This is the approach of Bonneau et al., among others.

# Cache Memory




$\langle x_i \rangle = \langle x_j \rangle$ are the higher bits of the data entry.

Data is pulled into cache based on the most-significant bits in its address.

# Cache Collisions Reveal Weaknesses

Let $T_E(K,P)$ be the encryption time for a plaintext $P$ using key $K$. Let $\bar{T}_E(K) = \frac{1}{n}\sum_{i=1}^{n} T_E(K,P_i)$, where $P_i$ is a random plaintext from $\{0|1\}^{128}$.

**Cache-Collision Assumption [?]**. For any pair of table lookups $i,j$, given a large enough number of *random* AES encryption that use the *same key*, $\bar{T}_E(K)$ will be lower when $\langle I_i \rangle = \langle I_j \rangle$ than when $\langle I_i \rangle \neq \langle I_j \rangle$

Note: The table lookup indices must be *independent* for random plaintexts.

# Cache Collisions (cont'd)

Let $a$ and $b$ be two memory addresses looked up in memory. Let $\langle a \rangle$ and $\langle b \rangle$ denote the MSBs of $a$ and $b$, respectively.

- Cache memory is organized into *lines*
  - MSB is mapped to the cache line index, LSB is mapped to the line (block) offset
- Reads on $a$ and $b$ cause a collision if $\langle a \rangle = \langle b \rangle$ (assuming other memory reads have not evicted (or invalidated) $a$ or $b$ from the cache.
- If $\langle a \rangle \neq \langle b \rangle$ then a cache collision *might* occur.
- We cannot say for certain whether or not the lower LSBs are equivalent...

# Attacks from Cache Collisions

That's it! We may now build an attack based on this result.

# LUT-Based Implementations

Let $X^i$ be the state of AES at round $i$. With the exception of $i = 10$, we have:

$$\begin{aligned} X^{i+1} = \{ & T_0[x_0^i] \oplus T_1[x_5^i] \oplus T_2[x_{10}^i] \oplus T_3[x_{15}^i] \oplus \{k_0^i, k_1^i, k_2^i, k_3^i\}, \\ & T_0[x_4^i] \oplus T_1[x_9^i] \oplus T_2[x_{14}^i] \oplus T_3[x_3^i] \oplus \{k_4^i, k_5^i, k_6^i, k_7^i\}, \\ & T_0[x_8^i] \oplus T_1[x_{13}^i] \oplus T_2[x_2^i] \oplus T_3[x_7^i] \oplus \{k_8^i, k_9^i, k_{10}^i, k_{11}^i\}, \\ & T_0[x_{12}^i] \oplus T_1[x_1^i] \oplus T_2[x_6^i] \oplus T_3[x_{11}^i] \oplus \{k_{12}^i, k_{13}^i, k_{14}^i, k_{15}^i\}\} \end{aligned}$$

# First Round

- First round: $x_i^0 = p_i \oplus k_i$
- With the T-box implementation, $x_0^0$, $x_4^0$, $x_8^0$, and $x_{12}^0$ are used as indices into $T_0$
- If we are looking for cache collisions, we must consider input bytes of the same T-box.

$$\begin{aligned} \langle x_i^0 \rangle = \langle x_j^0 \rangle &\Rightarrow \langle p_i \rangle \oplus \langle k_i \rangle = \langle p_j \rangle \oplus \langle k_j \rangle \\ &\Rightarrow \langle p_i \rangle \oplus \langle p_j \rangle = \langle k_i \rangle \oplus \langle k_j \rangle \end{aligned}$$

# First Round Attack Algorithm

ALGORITHM 1: FirstRoundAttack($N_s$)

$n \leftarrow 2^8 - 1$
$T \leftarrow \text{array}[0\dots n, 1\dots n, 0\dots n]$
**for** $i = 0 \rightarrow N_s$ **do**
 $P \leftarrow RandomBytes(16)$
 $start \leftarrow time()$
 $C \leftarrow E_K(P)$
 $end \leftarrow time()$
 $tt \leftarrow (start - end)$
 $T[i, j, \langle p_i \rangle \oplus \langle p_j \rangle] \leftarrow T[i, j, \langle p_i \rangle \oplus \langle p_j \rangle] + tt$
**end for**
$T[i, j, \langle p_i \rangle \oplus \langle p_j \rangle] \leftarrow T[i, j, \langle p_i \rangle \oplus \langle p_j \rangle] / N_s$
$mi, mj \leftarrow min(t)$ ▷ Use t-test if necessary
$\langle k_{mi} \rangle \oplus \langle k_{mj} \rangle \leftarrow \langle p_{mi} \rangle \oplus \langle p_{mj} \rangle$

# First Round Attack Algorithm (cont'd)

$$
\begin{aligned}
X^{i+1} = \{ & T_0[x_0^i] \oplus T_1[x_5^i] \oplus T_2[x_{10}^i] \oplus T_3[x_{15}^i] \oplus \{k_0^i, k_1^i, k_2^i, k_3^i\}, \\
& T_0[x_4^i] \oplus T_1[x_9^i] \oplus T_2[x_{14}^i] \oplus T_3[x_3^i] \oplus \{k_4^i, k_5^i, k_6^i, k_7^i\}, \\
& T_0[x_8^i] \oplus T_1[x_{13}^i] \oplus T_2[x_2^i] \oplus T_3[x_7^i] \oplus \{k_8^i, k_9^i, k_{10}^i, k_{11}^i\}, \\
& T_0[x_{12}^i] \oplus T_1[x_1^i] \oplus T_2[x_6^i] \oplus T_3[x_{11}^i] \oplus \{k_{12}^i, k_{13}^i, k_{14}^i, k_{15}^i\}\}
\end{aligned}
$$

# First Round Attack Algorithm (cont'd)

$$
\begin{aligned}
X^{i+1} = \{ & T_0[x_0^i] \oplus T_1[x_5^i] \oplus T_2[x_{10}^i] \oplus T_3[x_{15}^i] \oplus \{k_0^i, k_1^i, k_2^i, k_3^i\}, \\
& T_0[x_4^i] \oplus T_1[x_9^i] \oplus T_2[x_{14}^i] \oplus T_3[x_3^i] \oplus \{k_4^i, k_5^i, k_6^i, k_7^i\}, \\
& T_0[x_8^i] \oplus T_1[x_{13}^i] \oplus T_2[x_2^i] \oplus T_3[x_7^i] \oplus \{k_8^i, k_9^i, k_{10}^i, k_{11}^i\}, \\
& T_0[x_{12}^i] \oplus T_1[x_1^i] \oplus T_2[x_6^i] \oplus T_3[x_{11}^i] \oplus \{k_{12}^i, k_{13}^i, k_{14}^i, k_{15}^i\}\}
\end{aligned}
$$

# First Round Limitation

We only know that $\langle k_0 \rangle \oplus \langle k_4 \rangle = \Delta_1$, $\langle k_0 \rangle \oplus \langle k_8 \rangle = \Delta_2$, $\langle k_0 \rangle \oplus \langle k_{12} \rangle = \Delta_3$, $\langle k_4 \rangle \oplus \langle k_8 \rangle = \Delta_4$, $\langle k_4 \rangle \oplus \langle k_{12} \rangle = \Delta_5$, and $\langle k_8 \rangle \oplus \langle k_{12} \rangle = \Delta_6$.

Of course, there exists **18** other equations we can derive for $T_1$, $T_2$, and $T_3$.

We cannot determine the lower $\log_2$ bits of each key... What to do now?

# The Last Round

When $i = 10$, the lookup table is just the S-box $S$. At this point, the ciphertext $C$ is:

$$\begin{aligned} C = \{ & S[x_0^{10}] \oplus k_0^{10}, S[x_5^{10}] \oplus k_1^{10}, S[x_{10}^{10}] \oplus k_2^{10}, S[x_{15}^{10}] \oplus k_3^{10}, \\ & S[x_4^{10}] \oplus k_5^{10}, S[x_9^{10}] \oplus k_6^{10}, S[x_{14}^{10}] \oplus k_7^{10}, S[x_3^{10}] \oplus k_7^{10}, \\ & S[x_8^{10}] \oplus k_8^{10}, S[x_{13}^{10}] \oplus k_9^{10}, S[x_2^{10}] \oplus k_{10}^{10}, S[x_7^{10}] \oplus k_{11}^{10}, \\ & S[x_{12}^{10}] \oplus k_{12}^{10}, S[x_1^{10}] \oplus k_{13}^{10}, S[x_6^{10}] \oplus k_{14}^{10}, S[x_{11}^{10}] \oplus k_{15}^{10} \} \end{aligned}$$

# Final Round Collisions

Let $x_s$ and $x_t$ be two random bytes in the last round.

We will always have that $c_i = k_i^{10} \oplus S[x_s]$ and $c_j = k_j^{10} \oplus S[x_t]$. If $x_s = x_t$, then a collision *will usually* occur and $c_i = k_i^{10} \oplus \alpha$ and $c_j = k_j^{10} \oplus \alpha$.

Therefore, $c_i \oplus c_j = k_i^{10} \oplus k_j^{10}$

# Final Round Misses

What if $x_s \neq x_t$?

$c_i \oplus c_j \neq k_i^{10} \oplus k_j^{10} \Rightarrow$ two different values came out of the LUT!

The *S-box nonlinearity* means that the difference between $S[x_s]$ and $S[x_t]$ *does not* imply a similar difference between $x_s$ and $x_t$.

If $c_i \oplus c_j \neq k_i^{10} \oplus k_j^{10}$, then $x_s$ and $x_t$ are two *random* values.

# Final Round Attack Algorithm

ALGORITHM 2: LastRoundAttack($N_s$)

$diffs \leftarrow [0, \ldots, 2^n - 1, 0, \ldots, 2^n - 1]$
$T \leftarrow \text{array}[0 \ldots 2^n - 1, 0 \ldots 2^n - 1, 0 \ldots 2^n - 1]$
**for** $i = 0 \rightarrow N_s$ **do**
  $P \leftarrow RandomBytes(16)$
  $start \leftarrow time()$
  $C \leftarrow E_K(P)$
  $end \leftarrow time()$
  $tt \leftarrow (start - end)$
  **for** $i, j$ in $C$ **do**
    $T[i, j, C_i \oplus C_j] \leftarrow T[i, j, C_i \oplus C_j] + tt$
  **end for**
**end for**
$T[i, j, \Delta_{i,j}] \leftarrow T[i, j, \Delta_{i,j}] / N_s$
$\Delta'_{i,j} \leftarrow min(T, i, j)$ ▷ Use t-test if necessary **return** all $\Delta'_{i,j}$

# Final Round Attack Algorithm (cont'd)

With knowledge of $\Delta_{i,j}$ for all $i, j$ such that $\Delta_{i,j} = k_i^{10} \oplus k_j^{10}$ the attacker can now make informed guesses at the key

Thanks to Rijndael's invertible key schedule, $[k^{10}]$ can be reverted back to $[k^0]$.

Done.

# Timing Attack Countermeasures

- Keep?

# References

1. Joseph Bonneau and Ilya Mironov. Cache-collision timing attacks against AES. *Cryptographic Hardware and Embedded Systems CHES 2006*, Springer Berlin Heidelberg, (2006), 201-215.
2. Daniel J. Bernstein. Cache-timing attacks on AES. April 2005. `http://cr.yp.to/antiforgery/cachetiming-20050414.pdf`.