Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems

Firas Darwish

New York University Abu Dhabi

fbd2014@nyu.edu

ENGR-UH 4321: Introduction to Hardware Security
August 29, 2025

Introduction

Introduction

Cryptographic Algorithms take different amounts of time to process different inputs.

Introduction

Cryptographic Algorithms take different amounts of time to process different inputs. This is often due to:

Introduction

Cryptographic Algorithms take different amounts of time to process different inputs. This is often due to:

- Performance optimizations to bypass unnecessary operations.

Introduction

Cryptographic Algorithms take different amounts of time to process different inputs. This is often due to:

- Performance optimizations to bypass unnecessary operations.
- Branching and conditional statements.

Introduction

Cryptographic Algorithms take different amounts of time to process different inputs. This is often due to:

- Performance optimizations to bypass unnecessary operations.
- Branching and conditional statements.
- RAM cache hits.

Introduction

Cryptographic Algorithms take different amounts of time to process different inputs. This is often due to:

- Performance optimizations to bypass unnecessary operations.
- Branching and conditional statements.
- RAM cache hits.
- Processor instructions (multiplication and division) that run in non-fixed time.

Introduction

Cryptographic Algorithms take different amounts of time to process different inputs. This is often due to:

- Performance optimizations to bypass unnecessary operations.
- Branching and conditional statements.
- RAM cache hits.
- Processor instructions (multiplication and division) that run in non-fixed time.

$$\text{time to compute } f(x) \neq \text{ time to compute } f(y) \text{ if x} \neq \text{y}$$

Introduction

Cryptographic Algorithms take different amounts of time to process different inputs. This is often due to:

- Performance optimizations to bypass unnecessary operations.
- Branching and conditional statements.
- RAM cache hits.
- Processor instructions (multiplication and division) that run in non-fixed time.

$$\text{time to compute } f(x) \neq \text{ time to compute } f(y) \text{ if x} \neq \text{y}$$

This paper presents attacks which can exploit timing measurements from vulnerable systems to find an entire secret key.

Attack Set-Up

Attack Set-Up

Diffie-Hellman and RSA private-key operations consist of computing

$$R = y^x \mod n$$

where n is public and y can be found by an eavesdropper.

Attack Set-Up

Diffie-Hellman and RSA private-key operations consist of computing

$$R = y^x \mod n$$

where n is public and y can be found by an eavesdropper.

The attacker's goal is to find the secret key, x.

Attack Set-Up

Diffie-Hellman and RSA private-key operations consist of computing

$$R = y^x \mod n$$

where n is public and y can be found by an eavesdropper.

The attacker's goal is to find the secret key, x.

The assumptions of the attack are:

Attack Set-Up

Diffie-Hellman and RSA private-key operations consist of computing

$$R = y^x \mod n$$

where n is public and y can be found by an eavesdropper.

The attacker's goal is to find the secret key, x.

The assumptions of the attack are:

- The victim must compute R for several values of y.

Attack Set-Up

Diffie-Hellman and RSA private-key operations consist of computing

$$R = y^x \mod n$$

where n is public and y can be found by an eavesdropper.

The attacker's goal is to find the secret key, x.

The assumptions of the attack are:

- The victim must compute R for several values of y.
- y, n, and the computation time are known to the attacker.

Attack Set-Up

Diffie-Hellman and RSA private-key operations consist of computing

$$R = y^x \mod n$$

where n is public and y can be found by an eavesdropper.

The attacker's goal is to find the secret key, x.

The assumptions of the attack are:

- The victim must compute R for several values of y.
- y, n, and the computation time are known to the attacker.
- Secret key x is not ephemeral (it is constant throughout the attack).

Attack Set-Up

Diffie-Hellman and RSA private-key operations consist of computing

$$R = y^x \mod n$$

where n is public and y can be found by an eavesdropper.

The attacker's goal is to find the secret key, x.

The assumptions of the attack are:

- The victim must compute R for several values of y.
- y, n, and the computation time are known to the attacker.
- Secret key x is not ephemeral (it is constant throughout the attack).

While it is not our concern, we presume the necessary information and timing measurements might be obtained by passively eavesdropping on an interactive protocol.

Attack Set-Up

Continued

Attack Set-Up

Continued

We outline this attack using the simple modular exponentiation algorithm below which computes $R = y^x \mod n$, where x is w bits long:

Attack Set-Up
Continued

We outline this attack using the simple modular exponentiation algorithm below which computes $R = y^x \mod n$, where x is w bits long:

```
Let s_0 = 1 for k = 0 to w − 1 do
    if bit k of x is 1 then
        Let R_k = (s_k · y) mod n
    end
    else
        Let R_k = s_k
    end
    Let s_{k+1} = R_k^2 mod n
end
return R_{w−1};
```

Attack Set-Up

Continued

Attack Set-Up
Continued

The attack allows someone who knows exponent bits $0 \ldots (b-1)$ to find bit b.

Attack Set-Up
Continued

The attack allows someone who knows exponent bits $0 \ldots (b-1)$ to find bit b.

To obtain the entire exponent, start with b equal to 0 and repeat the attack until the entire exponent is known.

Attack Set-Up

Continued

The attack allows someone who knows exponent bits $0 \ldots (b-1)$ to find bit b.

To obtain the entire exponent, start with b equal to 0 and repeat the attack until the entire exponent is known.

- Normally $R_b = (s_b \cdot y) \mod n$ is fast but for certain pairs (s_b, y), the multiplication is slow that it takes longer than a typical whole exponentiation would.

Attack Set-Up

Continued

The attack allows someone who knows exponent bits $0 \ldots (b-1)$ to find bit b.

To obtain the entire exponent, start with b equal to 0 and repeat the attack until the entire exponent is known.

- Normally $R_b = (s_b \cdot y) \mod n$ is fast but for certain pairs (s_b, y), the multiplication is slow that it takes longer than a typical whole exponentiation would.
- Because the attacker has the pairs (s_b, y) that trigger the slow path, they can predict exactly when a particular multiple will be slow, if it is performed, thereby determining if the bit (of secret key) is 0 or 1.

Error Correction

Error Correction

If exponent bit b is guessed incorrectly, the predicted partial products (e.g. $s_{k+1}, s_{k+2}, \ldots$) and $R_{k \geq b}$ deviate randomly.

Error Correction

If exponent bit b is guessed incorrectly, the predicted partial products (e.g. $s_{k+1}, s_{k+2}, \ldots$) and $R_{k \geq b}$ deviate randomly.

- The actual measured times no longer line up with these wrong predictions in a consistent way.

Error Correction

If exponent bit b is guessed incorrectly, the predicted partial products (e.g. $s_{k+1}, s_{k+2}, \ldots$) and $R_{k \geq b}$ deviate randomly.

- The actual measured times no longer line up with these wrong predictions in a consistent way.

Error-Detection Property

After an incorrect bit guess, no more meaningful correlations are observed.

This can be used for *error correction* as the attacker can keep a small set of plausible guesses for the secret key bits found so far.

Error Correction

If exponent bit b is guessed incorrectly, the predicted partial products (e.g. $s_{k+1}, s_{k+2}, \ldots$) and $R_{k \geq b}$ deviate randomly.

- The actual measured times no longer line up with these wrong predictions in a consistent way.

Error-Detection Property

After an incorrect bit guess, no more meaningful correlations are observed.

This can be used for *error correction* as the attacker can keep a small set of plausible guesses for the secret key bits found so far.

As more timing data comes in:

Error Correction

If exponent bit b is guessed incorrectly, the predicted partial products (e.g. $s_{k+1}, s_{k+2}, \ldots$) and $R_{k \geq b}$ deviate randomly.

- The actual measured times no longer line up with these wrong predictions in a consistent way.

Error-Detection Property

After an incorrect bit guess, no more meaningful correlations are observed.

This can be used for *error correction* as the attacker can keep a small set of plausible guesses for the secret key bits found so far.

As more timing data comes in:

- The correct partial guess will continue to produce strong correlations (high likelihood).

Error Correction

If exponent bit b is guessed incorrectly, the predicted partial products (e.g. $s_{k+1}, s_{k+2}, \ldots$) and $R_{k \geq b}$ deviate randomly.

- The actual measured times no longer line up with these wrong predictions in a consistent way.

Error-Detection Property

After an incorrect bit guess, no more meaningful correlations are observed.

This can be used for *error correction* as the attacker can keep a small set of plausible guesses for the secret key bits found so far.

As more timing data comes in:

- The correct partial guess will continue to produce strong correlations (high likelihood).
- The incorrect guesses will not match well and see their likelihood collapse

Formalize the Attack

Formalize the Attack

Signal: timing variation due to the target exponent (secret) bit.

Formalize the Attack

Signal: timing variation due to the target exponent (secret) bit.
Noise: Measurement inaccuracies and timing variations due to unknown exponent bits.

Formalize the Attack

Signal: timing variation due to the target exponent (secret) bit.
Noise: Measurement inaccuracies and timing variations due to unknown exponent bits.

Formalize the Attack

Signal: timing variation due to the target exponent (secret) bit.
Noise: Measurement inaccuracies and timing variations due to unknown exponent bits.

Given j messages $y_0, y_1, \ldots, y_{j-1}$

Formalize the Attack

Signal: timing variation due to the target exponent (secret) bit.
Noise: Measurement inaccuracies and timing variations due to unknown exponent bits.

Given j messages $y_0, y_1, \ldots, y_{j-1}$ with corresponding timing measurements $T_0, T_1, \ldots, T_{j-1}$,

Formalize the Attack

Signal: timing variation due to the target exponent (secret) bit.
Noise: Measurement inaccuracies and timing variations due to unknown exponent bits.

Given j messages $y_0, y_1, \ldots, y_{j-1}$ with corresponding timing measurements $T_0, T_1, \ldots, T_{j-1}$, the probability that a guess x_b for the first b exponent bits is correct is proportional to

Formalize the Attack

Signal: timing variation due to the target exponent (secret) bit.
Noise: Measurement inaccuracies and timing variations due to unknown exponent bits.

Given j messages $y_0, y_1, \ldots, y_{j-1}$ with corresponding timing measurements $T_0, T_1, \ldots, T_{j-1}$, the probability that a guess x_b for the first b exponent bits is correct is proportional to

$$P(x_b) \propto \prod_{i=0}^{j-1} F(T_i - t(y_i, x_b))$$

where $t(y_i, x_b)$ is the amount of time required for the first b iterations of the $y_i^x \mod n$ computations using exponent bits x_b,

Formalize the Attack

Signal: timing variation due to the target exponent (secret) bit.
Noise: Measurement inaccuracies and timing variations due to unknown exponent bits.

Given j messages $y_0, y_1, \ldots, y_{j-1}$ with corresponding timing measurements $T_0, T_1, \ldots, T_{j-1}$, the probability that a guess x_b for the first b exponent bits is correct is proportional to

$$P(x_b) \propto \prod_{i=0}^{j-1} F(T_i - t(y_i, x_b))$$

where $t(y_i, x_b)$ is the amount of time required for the first b iterations of the $y_i^x \mod n$ computations using exponent bits x_b, and F is the expected probability distribution function of $T - t(y, x_b)$ over all y values and correct x_b (the *remaining* time).

Formalize the Attack

Continued

$$P(x_b) \propto \prod_{i=0}^{j-1} F(T_i - t(y_i, x_b))$$

Formalize the Attack

Continued

$$P(x_b) \propto \prod_{i=0}^{j-1} F(T_i - t(y_i, x_b))$$

- Proportionality as these are unnormalized probabilities.

Formalize the Attack
Continued

$$P(x_b) \propto \prod_{i=0}^{j-1} F(T_i - t(y_i, x_b))$$

- Proportionality as these are unnormalized probabilities.
- Kocker does not provide an implemention for finding $t(\cdot)$ but, in practice, t might be an approximation.

Formalize the Attack

Continued

$$P(x_b) \propto \prod_{i=0}^{j-1} F(T_i - t(y_i, x_b))$$

- Proportionality as these are unnormalized probabilities.
- Kocker does not provide an implemention for finding $t(\cdot)$ but, in practice, t might be an approximation.
- F is the model for ”What is the probability of seeing exactly that leftover time, if x_b is correct?”

Formalize the Attack

Continued

$$P(x_b) \propto \prod_{i=0}^{j-1} F(T_i - t(y_i, x_b))$$

- Proportionality as these are unnormalized probabilities.
- Kocker does not provide an implemention for finding $t(\cdot)$ but, in practice, t might be an approximation.
- F is the model for ”What is the probability of seeing exactly that leftover time, if x_b is correct?”
- Because we assume each measurement is (roughly) independent, we multiply these probabilities across $i = 0$ to $j - 1$.

Formalize the Attack
Continued

$$P(x_b) \propto \prod_{i=0}^{j-1} F(T_i - t(y_i, x_b))$$

- Proportionality as these are unnormalized probabilities.
- Kocker does not provide an implemention for finding $t(\cdot)$ but, in practice, t might be an approximation.
- F is the model for "What is the probability of seeing exactly that leftover time, if x_b is correct?"
- Because we assume each measurement is (roughly) independent, we multiply these probabilities across $i = 0$ to $j - 1$.
    - You multiply the individual likelihoods to get the joint likelihood for all j observations.

Formalize the Attack

Continued

Given a correct guess for x_{b-1}, there are two possible values for x_b.

Formalize the Attack

Continued

Given a correct guess for x_{b-1}, there are two possible values for x_b. The probability that x_b is correct and x_b' is incorrect can be found as

Formalize the Attack

Continued

Given a correct guess for x_{b-1}, there are two possible values for x_b. The probability that x_b is correct and x_b' is incorrect can be found as

$$\frac{\prod_{i=0}^{j-1} F(T_i - t(y_i, x_b))}{\prod_{i=0}^{j-1} F(T_i - t(y_i, x_b)) + \prod_{i=0}^{j-1} F(T_i - t(y_i, x_b'))}$$

Given a correct guess for x_{b-1}, there are two possible values for x_b. The probability that x_b is correct and x_b' is incorrect can be found as

$$\frac{\prod_{i=0}^{j-1} F(T_i - t(y_i, x_b))}{\prod_{i=0}^{j-1} F(T_i - t(y_i, x_b)) + \prod_{i=0}^{j-1} F(T_i - t(y_i, x_b'))}$$

If you notice, this step is just normalizing the probability we presented before.

Formalize the Attack

Continued

Given a correct guess for x_{b-1}, there are two possible values for x_b. The probability that x_b is correct and x_b' is incorrect can be found as

$$\frac{\prod_{i=0}^{j-1} F(T_i - t(y_i, x_b))}{\prod_{i=0}^{j-1} F(T_i - t(y_i, x_b)) + \prod_{i=0}^{j-1} F(T_i - t(y_i, x_b'))}$$

If you notice, this step is just normalizing the probability we presented before.

Likelihood Intractability

Directly computing $F(\cdot)$ is impractical, so we will need a simplification.

Simplifying the Attack

Simplifying the Attack

Each timing observation associated with one of j messages y consists of

Simplifying the Attack

Each timing observation associated with one of j messages y consists of

$$T = e + \sum_{i=0}^{w-1} t_i$$

where t_i is the time required for the multiplication and squaring steps for bit i

Simplifying the Attack

Each timing observation associated with one of j messages y consists of

$$T = e + \sum_{i=0}^{w-1} t_i$$

where t_i is the time required for the multiplication and squaring steps for bit i and e includes measurement error, loop overhead, caching, etc.

Simplifying the Attack

Each timing observation associated with one of j messages y consists of

$$T = e + \sum_{i=0}^{w-1} t_i$$

where t_i is the time required for the multiplication and squaring steps for bit i and e includes measurement error, loop overhead, caching, etc.

Given guess x_b, the attacker can find $\sum_{i=0}^{b-1} t_i$ for each sample y.

Simplifying the Attack

Each timing observation associated with one of j messages y consists of

$$T = e + \sum_{i=0}^{w-1} t_i$$

where t_i is the time required for the multiplication and squaring steps for bit i and e includes measurement error, loop overhead, caching, etc.

Given guess x_b, the attacker can find $\sum_{i=0}^{b-1} t_i$ for each sample y.

If x_b is correct, subtracting from T yields

Simplifying the Attack

Each timing observation associated with one of j messages y consists of

$$T = e + \sum_{i=0}^{w-1} t_i$$

where t_i is the time required for the multiplication and squaring steps for bit i and e includes measurement error, loop overhead, caching, etc.

Given guess x_b, the attacker can find $\sum_{i=0}^{b-1} t_i$ for each sample y.

If x_b is correct, subtracting from T yields

$$\left(e + \sum_{i=0}^{w-1} t_i\right) - \sum_{i=0}^{b-1} t_i = e + \sum_{i=b}^{w-1} t_i$$

Simplifying the Attack

Continued

Simplifying the Attack
Continued

We want to find the variance of $e + \sum_{i=b}^{w-1} t_i$.

Simplifying the Attack

Continued

We want to find the variance of $e + \sum_{i=b}^{w-1} t_i$. Based on our assumption of independence:

$$Var\left[e + \sum_{i=b}^{w-1} t_i\right] = Var\left[e\right] + (w - b)\, Var\left[t\right]$$

Simplifying the Attack
Continued

We want to find the variance of $e + \sum_{i=b}^{w-1} t_i$. Based on our assumption of independence:

$$Var\left[e + \sum_{i=b}^{w-1} t_i\right] = Var\left[e\right] + (w-b)\,Var\left[t\right]$$

What if only the first $c < b$ bits of our guess x_b is correct?

Simplifying the Attack

Continued

We want to find the variance of $e + \sum_{i=b}^{w-1} t_i$. Based on our assumption of independence:

$$Var\left[e + \sum_{i=b}^{w-1} t_i\right] = Var[e] + (w-b)\,Var[t]$$

What if only the first $c < b$ bits of our guess x_b is correct? Then the expected variance will be

$$Var[e] + (w-b+2c)\,Var[t]$$

Simplifying the Attack
Continued

We want to find the variance of $e + \sum_{i=b}^{w-1} t_i$. Based on our assumption of independence:

$$Var\left[e + \sum_{i=b}^{w-1} t_i\right] = Var\left[e\right] + (w - b)\, Var\left[t\right]$$

What if only the first $c < b$ bits of our guess x_b is correct? Then the expected variance will be

$$Var\left[e\right] + (w - b + 2c)\, Var\left[t\right]$$

Computing variances is easy and provides a good, statistical way to identify the correct exponent bit guess.

We want to find the variance of $e + \sum_{i=b}^{w-1} t_i$. Based on our assumption of independence:

$$Var\left[e + \sum_{i=b}^{w-1} t_i\right] = Var\left[e\right] + (w - b)\,Var\left[t\right]$$

What if only the first $c < b$ bits of our guess x_b is correct? Then the expected variance will be

$$Var\left[e\right] + (w - b + 2c)\,Var\left[t\right]$$

Computing variances is easy and provides a good, statistical way to identify the correct exponent bit guess. We just have to look at which guess x_b has the smaller expected variance!

Estimating the Number of Samples Required for the Attack

Estimating the Number of Samples Required for the Attack

Suppose an attacker has j accurate timing measurements

Estimating the Number of Samples Required for the Attack

Suppose an attacker has j accurate timing measurements and has two guesses for the first b bits of a w-bit key,

Estimating the Number of Samples Required for the Attack

Suppose an attacker has j accurate timing measurements and has two guesses for the first b bits of a w-bit key, one correct and the other incorrect with the first error at bit c.

Estimating the Number of Samples Required for the Attack

Suppose an attacker has j accurate timing measurements and has two guesses for the first b bits of a w-bit key, one correct and the other incorrect with the first error at bit c.

For each of these guesses, we adjust the timing measurements by $\sum_{i=0}^{b-1} t_i$ (from our estimated guess)

Estimating the Number of Samples Required for the Attack

Suppose an attacker has j accurate timing measurements and has two guesses for the first b bits of a w-bit key, one correct and the other incorrect with the first error at bit c.

For each of these guesses, we adjust the timing measurements by $\sum_{i=0}^{b-1} t_i$ (from our estimated guess) and then observe if its adjusted values have the smaller variance.

Estimating the Number of Samples Required for the Attack

Suppose an attacker has j accurate timing measurements and has two guesses for the first b bits of a w-bit key, one correct and the other incorrect with the first error at bit c.

For each of these guesses, we adjust the timing measurements by $\sum_{i=0}^{b-1} t_i$ (from our estimated guess) and then observe if its adjusted values have the smaller variance.

We present the probability that the variance of the incorrect guess is greater than the variance of the correct guess as

$$P\left(\sum_{i=0}^{j-1}\left(\sqrt{w-b}X_i+\sqrt{2(b-c)}Y_i\right)^2>\sum_{i=0}^{j-1}\left(\sqrt{w-b}X_i\right)^2\right)$$

Estimating the Number of Samples Required for the Attack

Suppose an attacker has j accurate timing measurements and has two guesses for the first b bits of a w-bit key, one correct and the other incorrect with the first error at bit c.

For each of these guesses, we adjust the timing measurements by $\sum_{i=0}^{b-1} t_i$ (from our estimated guess) and then observe if its adjusted values have the smaller variance.

We present the probability that the variance of the incorrect guess is greater than the variance of the correct guess as

$$P\left(\sum_{i=0}^{j-1}\left(\sqrt{w-b}X_i+\sqrt{2(b-c)}Y_i\right)^2 > \sum_{i=0}^{j-1}\left(\sqrt{w-b}X_i\right)^2\right)$$

We approximate the variance of t across j messages as a standard random variable.

Estimating the Number of Samples Required for the Attack

Continued

Estimating the Number of Samples Required for the Attack

Continued

A few steps produce the probability of a correct guess

Estimating the Number of Samples Required for the Attack

Continued

A few steps produce the probability of a correct guess

$$\Phi\left(\sqrt{\frac{j(b-c)}{2(w-b)}}\right)$$

where $\Phi(\cdot)$ is the area under the standard normal curve from $-\infty$ to x.

We note that:

Estimating the Number of Samples Required for the Attack

Continued

A few steps produce the probability of a correct guess

$$\Phi\left(\sqrt{\frac{j(b-c)}{2(w-b)}}\right)$$

where $\Phi(\cdot)$ is the area under the standard normal curve from $-\infty$ to x.
We note that:

1. the probability of a correct guess is proportional to the number of messages assessed (j).

Estimating the Number of Samples Required for the Attack

Continued

A few steps produce the probability of a correct guess

$$\Phi\left(\sqrt{\frac{j(b-c)}{2(w-b)}}\right)$$

where $\Phi(\cdot)$ is the area under the standard normal curve from $-\infty$ to x.
We note that:

1. the probability of a correct guess is proportional to the number of messages assessed (j).
2. The earlier the first error is in the incorrect guess, the higher the probability of a correct guess.

Estimating the Number of Samples Required for the Attack

Continued

A few steps produce the probability of a correct guess

$$\Phi\left(\sqrt{\frac{j(b-c)}{2(w-b)}}\right)$$

where $\Phi(\cdot)$ is the area under the standard normal curve from $-\infty$ to x.
We note that:

1. the probability of a correct guess is proportional to the number of messages assessed (j).
2. The earlier the first error is in the incorrect guess, the higher the probability of a correct guess.
3. The closer the length of the guess to the length of the key, the higher the probability of a correct guess.

Masking Timing Characteristics

Masking Timing Characteristics

- **Naïve solution**:

Masking Timing Characteristics

- **Naïve solution**: make all operations take exactly the same amount of time.

Masking Timing Characteristics

- **Naïve solution**: make all operations take exactly the same amount of time.
  - It is difficult to make software run in fixed time.

Masking Timing Characteristics

- **Naïve solution**: make all operations take exactly the same amount of time.
  - It is difficult to make software run in fixed time.
  - Performance optimizations cannot be used as all operations must take as long as the slowest operation.
- **Better solution**:

Masking Timing Characteristics

- **Naïve solution**: make all operations take exactly the same amount of time.
  - It is difficult to make software run in fixed time.
  - Performance optimizations cannot be used as all operations must take as long as the slowest operation.
- **Better solution**: Make timing measurements inaccurate so that the attack becomes unfeasible.

Masking Timing Characteristics

- **Naïve solution**: make all operations take exactly the same amount of time.
  - It is difficult to make software run in fixed time.
  - Performance optimizations cannot be used as all operations must take as long as the slowest operation.
- **Better solution**: Make timing measurements inaccurate so that the attack becomes unfeasible.
  - Adding random delays becomes noise with enough measurements from attacker.

Preventing an Attack

Blinding Signatures

Before computing the modular exponentiation operation,

Preventing an Attack
Blinding Signatures

Before computing the modular exponentiation operation, choose a random pair (v_i, v_f) such that $v_f^{-1} = v_i^x \mod n$.

Preventing an Attack

Blinding Signatures

Before computing the modular exponentiation operation, choose a random pair (v_i, v_f) such that $v_f^{-1} = v_i^x \mod n$.

Before the modular exponentiation operation, the input message should be multiplied by $v_i \mod n$,

Before computing the modular exponentiation operation, choose a random pair (v_i, v_f) such that $v_f^{-1} = v_i^x \mod n$.

Before the modular exponentiation operation, the input message should be multiplied by $v_i \mod n$, and afterward the result is corrected by multiplying with $v_f \mod n$,

Preventing an Attack

Blinding Signatures

Before computing the modular exponentiation operation, choose a random pair (v_i, v_f) such that $v_f^{-1} = v_i^x \mod n$.

Before the modular exponentiation operation, the input message should be multiplied by $v_i \mod n$, and afterward the result is corrected by multiplying with $v_f \mod n$, so the attacker has no useful knowledge about the input.

Before computing the modular exponentiation operation, choose a random pair (v_i, v_f) such that $v_f^{-1} = v_i^x \mod n$.

Before the modular exponentiation operation, the input message should be multiplied by $v_i \mod n$, and afterward the result is corrected by multiplying with $v_f \mod n$, so the attacker has no useful knowledge about the input.

The most that an attacker can learn is the general timing distribution for exponentiation operations.

Before computing the modular exponentiation operation, choose a random pair (v_i, v_f) such that $v_f^{-1} = v_i^x \mod n$.

Before the modular exponentiation operation, the input message should be multiplied by $v_i \mod n$, and afterward the result is corrected by multiplying with $v_f \mod n$, so the attacker has no useful knowledge about the input.

The most that an attacker can learn is the general timing distribution for exponentiation operations.

- *However, the distribution will reveal the average time per operation,*

Before computing the modular exponentiation operation, choose a random pair (v_i, v_f) such that $v_f^{-1} = v_i^x \mod n$.

Before the modular exponentiation operation, the input message should be multiplied by $v_i \mod n$, and afterward the result is corrected by multiplying with $v_f \mod n$, so the attacker has no useful knowledge about the input.

The most that an attacker can learn is the general timing distribution for exponentiation operations.

- *However, the distribution will reveal the average time per operation, which can be used to infer the Hamming weight of the exponent.*

References

Paul C. Kocher (1996)
Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems
Cryptography Research, Inc..

The End

Questions? Comments?