# Analysis of Man-in-the-Middle and Timing Attacks on Diffie-Hellman and RSA Protocols

Jiachen Pan
Student ID: 20233802046

Junlin Li
Student ID: 20233802058

Yueshen Wang
Student ID: 20233802018

Junrui Liang
Student ID: 20233802068

Abstract—Secure key exchange is really important for internet security today. The Diffie-Hellman (D-H) and RSA protocols are two common methods for establishing shared secrets over insecure networks like the internet. However, both protocols have known flaws that can be used to launch attacks. This report will focus on two major attacks: the Man-in-the-Middle (MitM) attack on D-H and the timing attack on RSA. We implemented both protocols from scratch in Python and then demonstrated how these attacks work in practice. We also implemented defensive techniques to protect against these attacks: adding RSA digital signatures to authenticate D-H key exchange, and using RSA blinding to prevent timing attacks. Our experimental results show that both attacks are practical and can be successfully executed, and the defenses we implemented can effectively stop them. We conclude by comparing these two different types of attacks and defenses, showing how protocol-level security (authentication) differs from implementation-level security (constant-time operations).

## I. Introduction

This report studies how two classic key exchange ideas can be attacked and how to defend them. We focus on Diffie-Hellman (D-H) and RSA. D-H helps two persons agree a shared secret over the internet. RSA is used for encryption and digital signatures.

These protocols are strong, but the basic versions have well-known gaps. Unauthenticated D-H is open to a man-in-the-middle (MitM) attacker. A simple RSA implementation can also leak information through its running time (a timing side-channel).

Our work is practical and hands-on:

1) We implement D-H and RSA in Python.
2) We show a MitM attack on unauthenticated D-H.
3) We measure a clear timing signal on a vulnerable RSA decryption.
4) We add two defenses: signatures to authenticate D-H, and blinding for RSA.

Section II gives short background. Section III explains our code. Section IV shows results from our runs. Section V compares the two attacks and defenses. Section VI concludes and suggests small next steps.

## II. Related Work

The weaknesses we study are well known in applied cryptography, and there is a lot of prior work on them.

### A. Foundational Protocols and Attack Vectors

The MitM attack on D-H is a classic example of why authentication is needed. The basic D-H protocol [1] does not verify who sent a message, so an attacker in the middle can relay and replace keys and stay hidden. The MITRE ATT&CK framework calls this T1557 (“Man-in-the-Middle”) [4], which matches the attack we simulate in dh_exchange.py.

Timing side-channel attacks, explained by Kocher in 1996 [3], are more tiny. They use information that leaks from the actual implementation behaviors, not from the math. In RSA, a simple square-and-multiply exponentiation takes a little bit longer when a private-key bit is 1 than 0. By timing many decryptions, an attacker can learn about the key.

Public CVEs show many real-world cases. For example, CVE-2018-5383 (the KNOB attack) is a MitM issue in Bluetooth. CVE-2021-3011 affected the cryptlib library and could leak information through timing. This shows that implementation issues keep appearing and need proper defenses.

### B. Broader Attack Surface and Defensive Frameworks

While we only show one attack per protocol, there are others. D-H can suffer from Logjam (weak or common primes). RSA can leak through other side-channels (e.g., power) or be broken by bad key generation.

To defend, the MITRE D3FEND framework lists countermeasures. Our D-H defense matches “Client-server Mutual Authentication” (D3-CSMA) [5]. Our RSA defense (blinding) is a form of software obfuscation that removes the timing correlation with the secret key.

Modern protocols such as TLS 1.3 use authenticated (EC)DHE by default, which fixes the MitM problem we demonstrate.

## III. Methods

This section explains what we built and how it works. We wrote everything in Python 3.9. For timing we use time.perf_counter_ns(). Our code is split into three files: rsa_utils.py (RSA helpers), dh_exchange.py (D-H simulation and attack), and rsa_attacks.py (timing attack experiment).

### A. Diffie-Hellman (D-H) Protocol

The D-H protocol is a key agreement scheme.

1) Mathematics: Two persons, Ana and Phara, agree on a large prime $p$ and a generator $g$. Ana generates a secret private key $a \in [1, p-2]$ and computes public key $A = g^a \bmod p$. Phara does the same, generating $b$ and $B = g^b \bmod p$. They exchange $A$ and $B$. Ana computes $S_A = B^a \bmod p = (g^b)^a \bmod p$. Phara computes $S_B = A^b \bmod p = (g^a)^b \bmod p$. Both result in the same shared secret $S = g^{ab} \bmod p$, as $S_A = S_B$.

2) Implementation: Our dh_exchange.py script implements this. For demonstration, it uses small, well-known parameters ($p = 23, g = 5$) to make the results verifiable by hand.

### B. Man-in-the-Middle Attack on D-H

The MitM attack exploits the D-H protocol's lack of authentication.

1) Logic: An attacker, Doomfist, positions herself between Ana and Phara. She performs 2 D-H exchanges:

- She catches Ana's $A$ and sends her own public key $M = g^m \bmod p$ to Phara.
- She catches Phara's $B$ and sends $M$ to Ana.

This results in two separate secret keys, $S_A = g^{am} \bmod p$ (known to Ana and Doomfist) and $S_B = g^{bm} \bmod p$ (known to Phara and Doomfist).

2) Implementation: Our simulate_mitm_attack() function in dh_exchange.py shows this. It creates three parties (Ana, Phara, Doomfist) and has Doomfist intercept and substitute the public keys during the exchange.

### C. D-H Defense: Authentication

We defend against MitM by adding a layer of authentication using RSA digital signatures.

1) Logic: This requires a Public Key Infrastructure (PKI), where Ana and Phara have shared, authentic copies of each other's RSA public key. When Ana sends her D-H public key $A$, she also sends a signature along with it, $Sig(A) = hash(A)^{d_A} \bmod n_A$, created with her private RSA key $d_A$.

2) Implementation: Our simulate_authenticated_exchange() function demonstrates this defense. Phara receives $(A, Sig(A))$ and checks it with Ana's public RSA key. Our script includes simple sign and verify functions. If the signature is valid, the key is authentic. Doomfist cannot create a valid signature because she does not have Ana's private RSA key.

### D. RSA Protocol

RSA is an asymmetric algorithm used for encryption and signatures.

1) Mathematics: Key generation involves:

- Selecting two large primes, $p$ and $q$.
- Computing the modulus $n = pq$.
- Computing the totient $\phi(n) = (p-1)(q-1)$.
- Choosing a public exponent $e$ (we used 65537, which is commonly used) coprime to $\phi(n)$.
- Calculating the private exponent $d$ using the modular inverse such that $ed \equiv 1 \pmod{\phi(n)}$.

The public key is $(e, n)$ and the private key is $(d, n)$. Encryption: $C = M^e \bmod n$. Decryption: $M = C^d \bmod n$.

2) Implementation: Our rsa_utils.py script provides functions for is_prime (a Miller-Rabin test) and mod_inverse to build a generate_keypair function. We used Python's built-in pow(e, -1, phi) for the modular inverse because it is efficient and reliable.

### E. Timing Attack on RSA

This attack exploits a non-constant-time decryption function.

1) Vulnerability: A basic square-and-multiply algorithm for decryption is often not "constant-time." The operation only performs a multiplication step when a bit in the private key $d$ is a '1'. This creates a small time difference that can be measured. Our vulnerable_decrypt function in rsa_attacks.py is an example of this.

2) Attack Logic: The attacker sends loads of random ciphertexts $C_i$ to the server and measures the precise time $T_i$ for each decryption. By performing analysis on the set of timings $\{T_i\}$, the attacker can get to know about the time distributions for '0' bits vs. '1' bits, and finally recover $d$.

### F. RSA Defense: Blinding

Blinding is a mathematical strategy that randomizes the input to the vulnerable function.

1) Logic: Before decryption, the server (holder of $d$) generates a random number $r$ (the blinding factor). It then "blinds" the ciphertext $C$:

$$C' = C \cdot r^e \bmod n \tag{1}$$

It decrypts $C'$:

$$M' = (C')^d \bmod n = (C \cdot r^e)^d \bmod n = (M \cdot r) \bmod n \tag{2}$$

Finally, it "unblinds" the result to get $M$:

$$M = M' \cdot r^{-1} \bmod n \tag{3}$$

where $r^{-1}$ is the modular inverse of $r$ modulo $n$.

2) Implementation: Our blinded_decrypt function in rsa_attacks.py implements this. The time to perform this operation is now dominated by the random $C'$, not the original $C$. The correlation between the timing and the bits of $d$ is broken.

### G. Experimental Setup

All implementations were developed using Python 3.9, executed on macOS. The D-H simulation used small parameters ($p = 23, g = 5$) for clarity and easy verification. The RSA timing attack used 512-bit keys (two 256-bit primes) to ensure the computations were measurable while maintaining reasonable execution time. High-resolution timing was performed using time.perf_counter_ns() over 5,000 trials.

## IV. Results

This section presents the data generated from executing our Python scripts.

### A. D-H Normal vs. MitM Exchange

Our dh_exchange.py script first simulates a normal D-H exchange, then simulates the MitM attack. The output from our execution is captured in Table I.

TABLE I
Comparison of Shared Keys in D-H Scenarios

| Scenario | $a$ | $b$ | $m$ | Ana's Secret | Phara's Secret | Doomfist's Secrets |
|---|---|---|---|---|---|---|
| Normal | 15 | 18 | N/A | 8 | 8 | N/A |
| MitM | 21 | 6 | 12 | 9 | 8 | (9, 8) |

From Table I, in the normal run both sides agree on the same secret (8). In the MitM run, Ana ends up with 9 (shared with Doomfist) and Phara ends up with 8 (also shared with Doomfist). Doomfist knows both, so she can read/modify everything even though both victims think they are secure.

### B. D-H Defense Validation

We then ran the simulate_authenticated_exchange() function. In this test, Doomfist catches Ana's message $(A, Sig(A))$ and tries to replace it with her own $(M, Fake\_Sig(M))$. Phara's client, upon receiving this, runs the verify function. Because Doomfist cannot fake Ana's signature, this verification fails. Our script printed the error message: Phara: Signature verification FAILED. Attack detected. Similarly, Ana also detected the attack. The defense was 100% effective in stopping the attack.

After detecting Doomfist's faked messages, Ana and Phara then proceed to exchange their real, signed keys. Both successfully verify each other's authentic signatures, and they compute the same shared secret (12 in our run), confirming that SUCCESS: Defense worked. MitM was prevented. The authentication layer successfully defeats the MitM attack.

### C. RSA Timing Attack Analysis

We ran our run_timing_experiment() script from rsa_attacks.py with 5,000 trials against the vulnerable_decrypt function. The key was 512 bits.

Our results showed a very high variance in execution time, which is the signature of a timing leak. The key results are shown in Table II.

This huge difference is caused by the non-constant-time square-and-multiply loop. When a private-key bit is 1, the code does an extra multiply and runs a bit longer. Over many trials this creates a signal an attacker can use, as Kocher explained [3].

TABLE II
Timing Statistics for Vulnerable RSA Decryption

| Metric | Value |
|---|---|
| Average Time | 1,974,350.31 ns (≈1.97 ms) |
| Min Time | 1,893,958.00 ns |
| Max Time | 26,649,875.00 ns |
| Time Variation (Max - Min) | 24,755,917.00 ns (≈24.76 ms) |

### D. RSA Blinding Defense Validation

We then ran the same timing experiment against our blinded_decrypt function. The results are shown in Table III.

TABLE III
Timing Statistics for Blinded RSA Decryption

| Metric | Value |
|---|---|
| Average Time | 2,051,034.79 ns (≈2.05 ms) |
| Min Time | 2,014,042.00 ns |
| Max Time | 2,595,958.00 ns |
| Time Variation (Max - Min) | 581,916.00 ns (≈0.58 ms) |

The results showed a clear and successful mitigation of the timing leak. The average time is slightly higher, which is expected due to the extra blinding operations. The key result is the dramatic reduction in time variation (from ∼24.76 ms to ∼0.58 ms, a reduction of about 42×). With blinding, the timing no longer correlates with the private key bits. The remaining variation is random noise that is not useful for an attacker.

## V. Discussion

This project includes a comparison of two fundamentally different types of attacks and their corresponding defenses.

### A. Comparison of Attack Methods (MitM vs. Timing)

The MitM attack on D-H and the timing attack on RSA differ significantly in their target, required conditions, and impact (Table IV).

TABLE IV
Comparison of Attack Methods

| Aspect | MitM Attack | Timing Attack |
|---|---|---|
| Target | Protocol (no authentication) | Code implementation (side-channel) |
| Activeness | Active (intercept/modify traffic) | Mostly passive (send requests, measure time) |
| Impact | Breaks a session | Leaks private key (breaks many future sessions) |

### B. Comparison of Defensive Methods (Authentication vs. Blinding)

The defenses are similarly different in their approach, each tailored to the specific attack (Table V).

TABLE V
Comparison of Defensive Methods

| Aspect | Authentication | Blinding |
|---|---|---|
| Mechanism | Add signatures to check identity (fixes protocol gap) | Blind input so timing doesn't reveal key bits (fixes implementation leak) |
| Performance | Needs sign and verify operations | Needs generating $r$, computing $r^e$, and inverse |
| Scope | General idea used in many protocols | Specific to public-key operations with timing leaks |

## VI. Conclusion and Future Work

To sum up, we built simple versions of D-H and RSA, showed two classic attacks (MitM and timing), and added two standard defenses (signatures and blinding). Unauthenticated D-H is not safe against an active attacker, and a non-constant-time RSA decrypt leaks timing information.

With our defenses, both issues are fixed in our tests. Signatures stop the MitM, and blinding removes the timing signal.

The main lesson is: protocols need authentication, and implementations need constant-time behavior (or blinding). Missing either one can break securit of the 'theoritical secure' system.

For future work:

1) Add better timing analysis and plots (e.g., matplotlib).
2) Try a lower-level language to reduce Python runtime noise like C++.

## Authors' Contributions

- Jiachen Pan (20233802046): Implemented Diffie–Hellman protocol, Man-in-the-Middle attack, and authenticated D–H defense (code/dh_exchange.py). Wrote Methods (D–H) and Results (D–H) sections.
- Junlin Li (20233802058): Implemented RSA helpers and key generation (code/rsa_utils.py). Wrote Methods (RSA) and part of Related Work.
- Yueshen Wang (20233802018): Implemented RSA timing attack and blinding defense (code/rsa_attacks.py). Ran experiments and wrote Results (RSA) section.
- Junrui Liang (20233802068): Wrote Abstract, Introduction, Discussion, and Conclusion. Converted the paper to IEEE LaTeX format and prepared Table 1.

All authors reviewed and approved the final paper. Each author contributed to code testing and small fixes during integration.

## References

[1] W. Diffie and M. E. Hellman, "New directions in cryptography," IEEE Transactions on Information Theory, vol. 22, no. 6, pp. 644-654, Nov. 1976.
[2] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," Communications of the ACM, vol. 21, no. 2, pp. 120-126, Feb. 1978.
[3] P. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," in Advances in Cryptology — CRYPTO '96, Berlin, Heidelberg, 1996, pp. 104–113.
[4] MITRE. ATT&CK T1557: Man-in-the-Middle. [Online]. Available: https://attack.mitre.org/techniques/T1557/
[5] MITRE. D3FEND D3-CSMA: Client-server Mutual Authentication. [Online]. Available: https://d3fend.mitre.org/technique/d3fend/D3-CSMA/