Design and Evaluation of Transaction Intent Verification for Crypto Wallet Approval Phishing Mitigation Using EIP-712 Typed Data Signatures

Sarah Alwa Neguita Surbakti - 18223023
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: sarahsrbkt2765@gmail.com , 18223023@std.stei.itb.ac.id

***Abstract*—This research presents the design and evaluation of a transaction intent verification system for mitigating crypto wallet approval phishing. Approval phishing may occur when users sign or approve transaction data that does not match their actual intention. To address this issue, this study implements a graphical prototype called IntentGuard Wallet using EIP-712 typed data signatures. The system allows users to connect a crypto wallet, create transaction intent data, sign the data through MetaMask, and verify whether the signature still matches the current transaction information. The transaction intent includes wallet address, spender address, token address, amount, chain ID, nonce, deadline, and verifying contract. Several experimental scenarios were tested, including valid signing, spender tampering, token tampering, amount modification, chain ID modification, expired deadline, and replay nonce attempt. The results show that the system can detect changes in signed transaction data and reject reused nonce attempts. This indicates that transaction intent verification using EIP-712 can help users identify suspicious approval changes before trusting a wallet signature.**

Keywords—component; EIP-712, Transaction Intent, Typed Data Signature, Crypto Wallet, Approval Phishing, Digital Signature, Blockchain Security

I. INTRODUCTION

Crypto wallets are widely used to interact with blockchain applications, such as token transfers, decentralized exchanges, and other decentralized finance services. In many cases, users are asked to sign messages or approve token permissions before a transaction can be executed. However, this process can become a security risk when users do not clearly understand what they are signing. One common attack is approval phishing, where a user is tricked into approving access to tokens or signing data that does not match the user's actual intention.

Approval phishing is dangerous because the signed or approved data may allow an attacker to move tokens, increase token allowance, or use a different spender address than expected. Many users only see a wallet confirmation pop-up and approve it without checking the details carefully. This problem shows the need for a clearer verification process before a wallet signature is trusted. A transaction intent verification system can help by comparing the data that was signed with the transaction data that is currently being used.

This report presents IntentGuard Wallet, a graphical prototype that uses EIP-712 typed data signatures to create and verify transaction intent. The system allows users to connect a wallet, input transaction intent data, sign the data through MetaMask, and verify whether the signature still matches the current data. The intent data includes wallet address, spender address, token address, amount, chain ID, nonce, deadline, and verifying contract. This report focuses on three objectives: (1) designing a transaction intent schema using EIP-712 typed data, (2) implementing a GUI prototype for signing and verification, and (3) evaluating the system through several attack scenarios, including spender tampering, token tampering, amount modification, chain ID modification, expired deadline, and replay nonce attempt.

II. BACKGROUNDS AND RELATED WORKS

A. Crypto Wallet and Digital Signature

A crypto wallet is used to manage blockchain accounts and interact with decentralized applications. In Ethereum-based systems, a wallet can be used to send transactions, approve token spending, and sign messages. A digital signature proves that a message or transaction was approved by the owner of a private key. The public address can later be used to verify whether the signature was made by the expected wallet owner.

In wallet-based applications, signing is often used before a user performs an action. For example, a user may sign a message to log in, approve token usage, or confirm a transaction intent. However, this process can be risky when the signed data is difficult to read or does not clearly show what the user is approving. If users only trust the wallet pop-up without checking the details, they may approve malicious data.

B. Token Approval and Approval Phishing

Token approval is a common mechanism in Ethereum applications. In an ERC-20 token, the owner can approve another address, called a spender, to use a certain amount of tokens on behalf of the owner [1]. This mechanism is useful for decentralized exchanges and other smart contract applications because the contract needs permission to move tokens during a transaction.

However, token approval can also be abused. In approval phishing, an attacker tricks a user into approving a malicious spender address or signing data that gives access to tokens. The user may believe that the approval is harmless, but the approved spender can later use the permission to move tokens. This attack is dangerous because the user technically approved the permission with their own wallet. Therefore, the main issue is not only whether the signature is valid, but also whether the signed data matches the user's real transaction intent.

C. EIP-712 Typed Data Signature

EIP-712 is a standard for signing typed structured data in Ethereum [2]. Unlike a plain message signature, EIP-712 allows data to be organized into clear fields, such as address, amount, chain ID, nonce, and deadline. This makes the signed message easier to display, understand, and verify.

An EIP-712 signature usually consists of three main parts: domain, types, and message. The domain defines the signing context, such as application name, version, chain ID, and verifying contract. The types define the data structure. The message contains the actual values that will be signed. In this research, the transaction intent is represented as a typed data structure with the following fields: wallet, spender, token, amount, chain ID, nonce, deadline, and verifying contract.

The simplified transaction intent structure used in this project is shown below:

$$TransactionIntent = \{wallet, spender, token, amount, chainId, nonce, deadline\}$$

The domain data is also important because it separates one application or contract context from another. For example, a signature created for one chain or one verifying contract should not be trusted in a different context. This helps reduce the risk of signature reuse across different environments.

D. Transaction Intent Verification

Transaction intent refers to the transaction details that the user actually wants to approve. In this project, transaction intent includes the wallet address, spender address, token address, amount, chain ID, nonce, deadline, and verifying contract. These fields were selected because they represent important information in an approval process. If one of these fields changes after signing, the signature should no longer be considered valid for the current transaction.

The verification process checks whether the current transaction data still matches the data that was signed. First, the system recovers the signer address from the EIP-712 signature. Second, the recovered signer is compared with the wallet address in the intent. Third, the system checks whether the deadline has expired. Fourth, the nonce is checked to prevent replay attempts. If one of these checks fails, the system returns INVALID.

This approach is useful because it does not only check whether a signature exists. It also checks whether the signature is still consistent with the current transaction intent. This is important for approval phishing mitigation, because an attacker may try to change the spender, token, amount, chain ID, or reuse a valid signature.

E. Related Works

Previous wallet security approaches often focus on warning users about risky transactions, showing transaction simulations, or improving wallet confirmation screens. These approaches are useful, but users may still approve malicious data if the displayed information is unclear or if important fields are hidden. EIP-712 improves this situation by allowing applications to present structured data before signing [2].

Several wallet libraries and tools also support typed data signing and verification. MetaMask supports EIP-712 signing through typed data requests, while ethers.js provides functions for creating and verifying typed data signatures [3], [4]. However, EIP-712 alone does not automatically prevent all attacks. Additional checks such as nonce validation, deadline validation, and transaction parameter matching are still needed.

This research builds on those ideas by implementing a simple graphical prototype called IntentGuard Wallet. The prototype does not send real transactions to the blockchain. Instead, it focuses on signing, verification, and experimental attack simulation. The contribution of this work is the design and evaluation of a transaction intent verification flow that detects spender tampering, token tampering, amount modification, chain ID modification, expired deadline, and replay nonce attempts.

III. METHODOLOGY

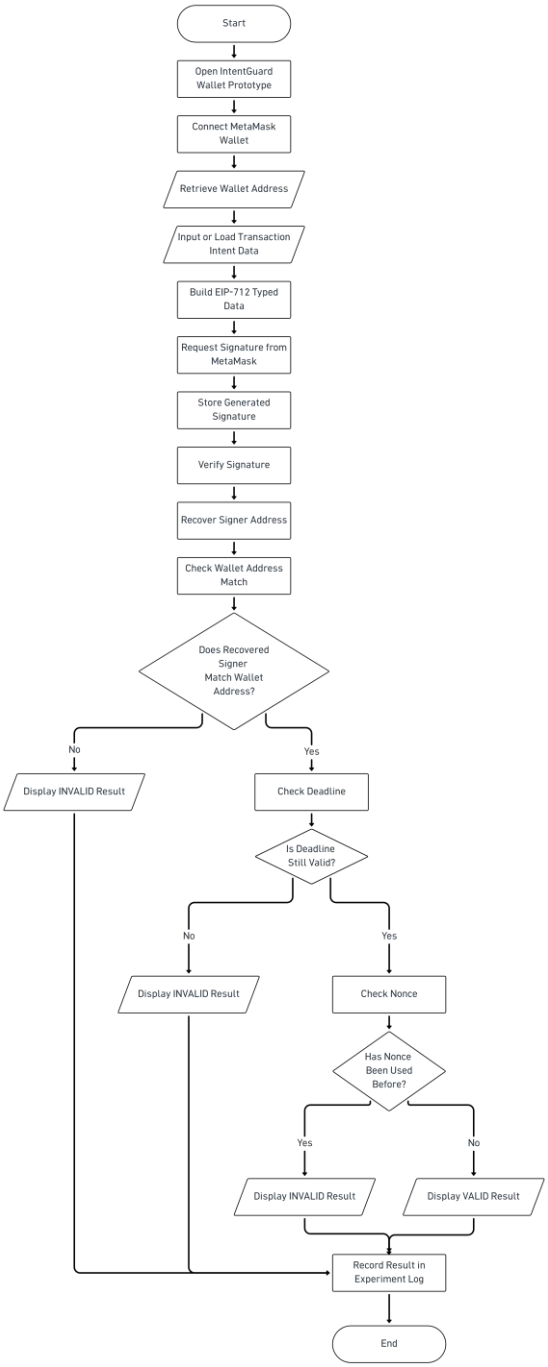
A. Prototype Implementation Details

The system was implemented as a graphical web-based prototype called IntentGuard Wallet. The prototype was developed using React and Vite for the user interface, ethers.js for wallet interaction and signature verification, and MetaMask as the crypto wallet provider. The main goal of the system is to let users create a transaction intent, sign it using EIP-712 typed data signatures, and verify whether the signed data still matches the current transaction data.

The prototype does not send real transactions to the blockchain. Instead, it focuses on the signing and verification process. This design was chosen because the main problem in approval phishing often happens before the transaction is executed, when users approve or sign data without fully checking the details. By separating the signing and verification process, the system can show how transaction intent verification can detect data changes before the user trusts the approval.

The application consists of several main components. The wallet connection component connects the user's MetaMask wallet and retrieves the wallet address. The transaction intent form stores the approval data that will be signed. The signing section requests an EIP-712 signature from the wallet. The verification section recovers the signer address and checks whether the current data is still valid. The attack simulation section changes selected fields after signing to test whether the system can detect tampering. The experiment log records the result of each verification attempt. Fig. 1 shows the general workflow of the IntentGuard Wallet prototype.

Fig. 1. Workflow of The IntentGuard Wallet Prototype



B. *Transaction Intent Schema Design*

The transaction intent schema was designed to represent the most important information in a crypto wallet approval process. The schema contains wallet address, spender address, token address, amount, chain ID, nonce, deadline, and verifying contract. These fields were selected because each of them has a direct impact on whether an approval is safe or risky.

The wallet address represents the user who signs the data. The spender address represents the account or contract that will receive permission to use the token. The token address identifies the asset involved in the approval. The amount specifies the value that is approved. The chain ID prevents the signed data from being trusted on a different blockchain network. The nonce is used to detect replay attempts. The deadline limits the validity period of the signature. The verifying contract defines the contract context where the signature should be valid.

The simplified transaction intent structure used in this prototype is shown below:

$$TransactionIntent = \{wallet, spender, token, amount, chainId, nonce, deadline\}$$

The EIP-712 domain used in the prototype contains the application name, version, chain ID, and verifying contract. This domain separates the signing context from other applications or contracts. If the chain ID or verifying contract changes after signing, the verification result should become invalid.

C. *Signing Process*

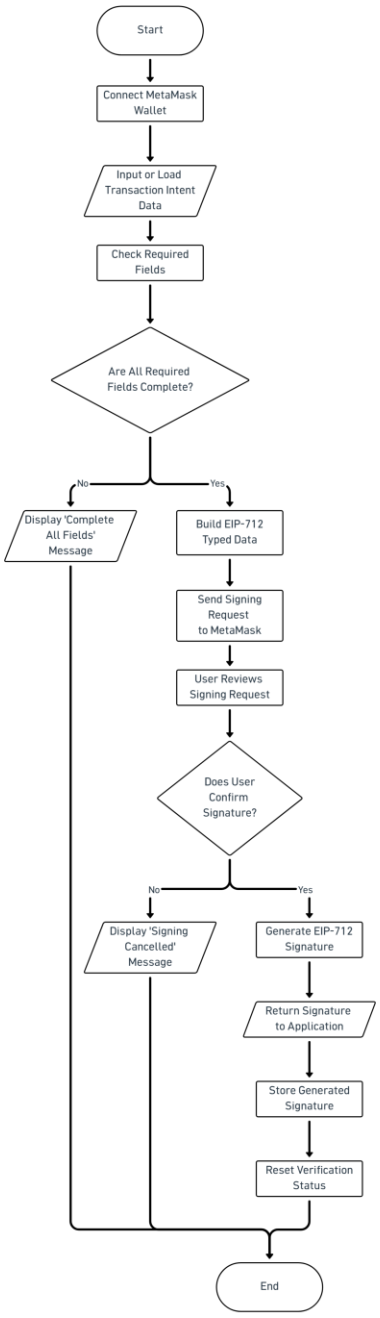
The signing process starts when the user connects a wallet and fills in the transaction intent form. The user can also load sample data to make testing easier. After the required fields are filled, the system builds the EIP-712 typed data using the transaction intent schema and domain data.

The signing request is then sent to MetaMask. The user reviews the signing request and confirms it through the wallet. If the signing process is successful, MetaMask returns a signature. This signature is stored in the application and later used for verification. After a new signature is created, the verification status is reset because the current signature has not been checked yet.

The signing process can be summarized as follows:

1. User connects the wallet.
2. User fills or loads transaction intent data.
3. System builds EIP-712 typed data.
4. User signs the typed data through MetaMask.
5. System stores the generated signature.

Fig. 2. Signing Flow Used in The Prototype



D. Verification Process

The verification process checks whether the signature still matches the current transaction intent. First, the system rebuilds the EIP-712 typed data from the current form values. Then, the system uses the signature to recover the signer address. The recovered signer is compared with the wallet address stored in the transaction intent.

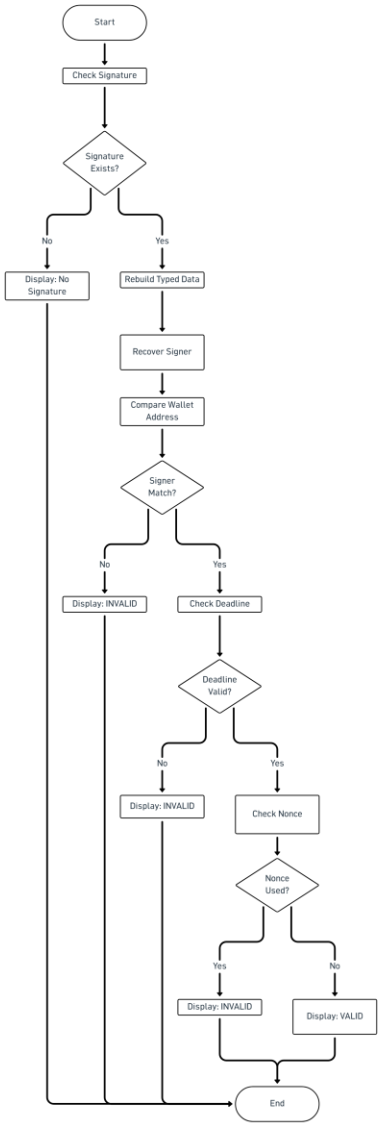
After the signer is checked, the system also checks the deadline and nonce. If the deadline is already passed, the signature is marked as invalid. If the nonce has already been used, the signature is also marked as invalid. These checks are important because a valid signature should not be trusted forever or reused multiple times.

The verification process returns VALID only if all conditions are satisfied. The signature must match the current data, the recovered signer must match the wallet address, the deadline must still be valid, and the nonce must not have been used before. If one condition fails, the system returns INVALID.

The verification process can be summarized as follows:

1. System checks whether a signature exists.
2. System rebuilds the EIP-712 typed data from the current intent.
3. System recovers the signer address from the signature.
4. System compares the recovered signer with the wallet address.
5. System checks the deadline.
6. System checks whether the nonce has been used.
7. System returns VALID or INVALID.

Fig. 3. Signature Verification Flow



E. *Attack Simulation Design*

Several attack simulation buttons were added to evaluate whether the system can detect changes after signing. These simulations represent common approval phishing risks where the signed data is modified or reused. The tested scenarios include spender tampering, token tampering, amount tampering, chain ID modification, expired deadline, and replay nonce attempt.

In the spender tampering scenario, the spender address is changed to another address after the data has been signed. In the token tampering scenario, the token address is changed. In the amount tampering scenario, the approved amount is increased. In the chain ID modification scenario, the network identifier is changed. In the expired deadline scenario, the deadline is changed to a past timestamp. In the replay nonce scenario, the same nonce is used again after a valid verification.

Each scenario is tested by first signing valid transaction intent data, then modifying one condition, and finally running the verification process again. The expected result is that the first valid signing should return VALID, while all tampering and replay scenarios should return INVALID.

F. *Experimental Design*

The experiment was designed to evaluate whether the prototype can distinguish valid transaction intent from modified or reused data. The experiment uses a fixed sample transaction intent as the baseline data. The baseline data contains a wallet address from MetaMask, a sample spender address, a sample token address, amount value, chain ID, nonce, deadline, and verifying contract.

The testing process was carried out using the following steps:

1. Connect MetaMask wallet.
2. Load sample transaction intent data.
3. Sign the transaction intent using EIP-712.
4. Verify the original signed data.
5. Run one attack simulation.
6. Verify the modified data.
7. Record the result in the experiment log.

The expected result for the original signed data is VALID. The expected result for each attack scenario is INVALID. The experiment log records the scenario, verification result, and recovered signer address. This log is used to support the evaluation section of this report.

TABLE I. EXPERIMENTAL SCENARIOS

No.	Scenario	Expected Result
1	Valid transaction intent	VALID
2	Spender address changed after signing	INVALID
3	Token address changed after signing	INVALID
4	Amount changed after signing	INVALID
5	Chain ID changed after signing	INVALID
6	Deadline expired	INVALID
7	Nonce reused	INVALID

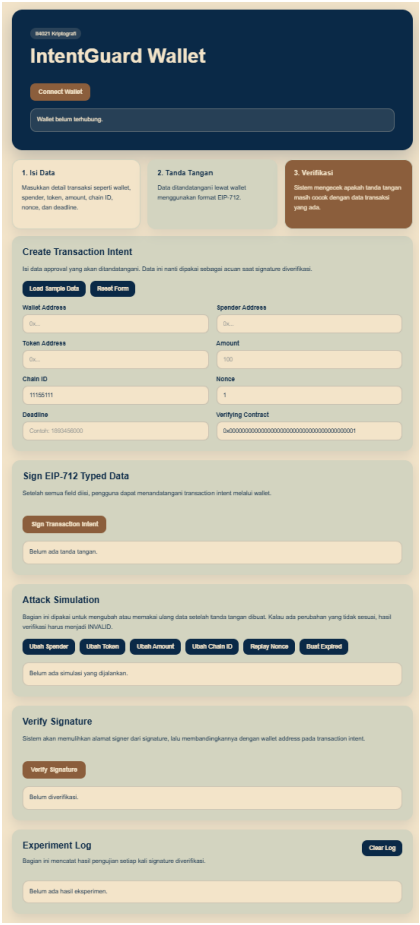
IV. RESULT AND ANALYSIS

A. *Prototype Result*

The IntentGuard Wallet prototype was successfully implemented as a web-based graphical application. The application allows users to connect a MetaMask wallet, create transaction intent data, sign the data using EIP-712 typed data signatures, and verify the signature. It also provides attack simulation buttons to test whether signed data can still be trusted after important fields are changed. The experiment log records each verification result so the testing process can be reviewed clearly.

The main interface contains wallet connection, transaction intent form, signing section, attack simulation section, verification result, and experiment log. The wallet connection section retrieves the user address from MetaMask. The form stores approval data such as spender, token, amount, chain ID, nonce, deadline, and verifying contract. Fig. 6 shows the main interface of the IntentGuard Wallet prototype.

Fig. 4. Main interface of IntentGuard Wallet

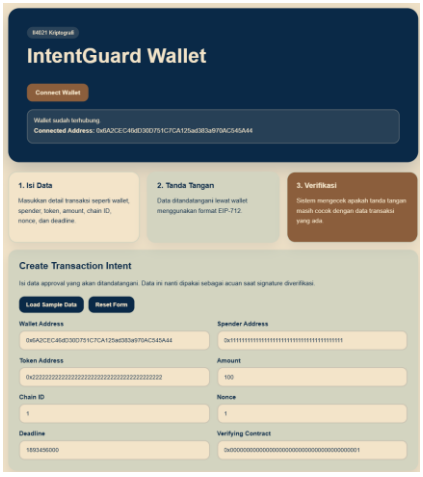


B. *Valid Transaction Intent Testing*

The first test was performed using valid transaction intent data. In this scenario, the user connected a MetaMask wallet and loaded the sample transaction intent data into the form. The

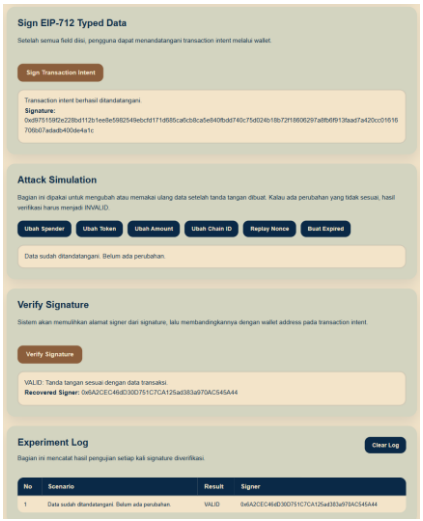
sample data included the wallet address, spender address, token address, amount, chain ID, nonce, deadline, and verifying contract. Fig. 5 shows the sample transaction intent data that was loaded in the prototype.

Fig. 5. Sample transaction intent data loaded in the prototype



After the data was loaded, the user signed the transaction intent through MetaMask and then verified the generated signature. The system returned VALID because the transaction intent used during verification was still the same as the data used during signing. The recovered signer address also matched the wallet address stored in the transaction intent. Fig. 6 shows the valid verification result and the experiment log recorded by the prototype.

Fig. 6. Verification result and experiment log for valid transaction intent



C. *Tampering Scenario Testing*

The next tests were performed by modifying transaction intent data after the signature had been created. This was done to simulate approval phishing attempts where an attacker changes important approval information after the user signs the original data. The tested tampering scenarios include changing the spender address, changing the token address, increasing the amount, and changing the chain ID. After each change, the same signature was verified again using the modified data.

The system returned INVALID for all tampering scenarios. This happened because an EIP-712 signature is linked to the exact structured data that was signed. If one field is changed, the current typed data no longer matches the original signed data. Fig. 7 shows an example of an INVALID result after the spender address was changed.Fig. 7. Invalid verification result after transaction intent tampering.

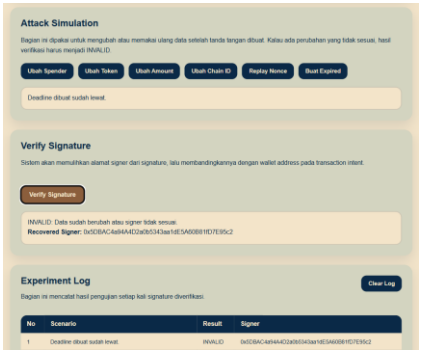
Fig. 7. Verification result and experiment log for valid transaction intent

D. *Deadline Expiration Testing*

The deadline expiration test was performed by changing the deadline value to a past timestamp. This scenario represents a case where an old signature should no longer be accepted. After the deadline was changed, the system returned INVALID. This result shows that the deadline field can limit the validity period of a signature.

Without a deadline check, an old signature may still be accepted even when it should no longer be trusted. By adding deadline validation, the system can reject signatures that have expired. This makes the signed approval data safer because it cannot be trusted forever. The result also shows that deadline validation is an important part of transaction intent verification.

Fig. 8. Invalid verification result after deadline expiration



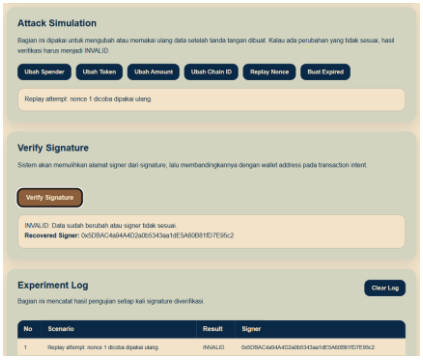
E. *Replay Nonce Testing*

The replay nonce test was performed to check whether the same signature could be reused. In the first verification, the valid

signature was accepted and the nonce was recorded as used. In the second verification, the same nonce was used again through the replay nonce scenario. The system returned INVALID because the nonce had already been recorded before.

Fig. 9 shows the replay nonce attempt recorded in the experiment log. This result shows that nonce validation is important for replay protection. EIP-712 provides structured data signing, but replay prevention still needs to be handled by the application or smart contract logic. In this prototype, the used nonce is stored during runtime and checked during every verification process.Fig. 8. Experiment log for valid, tampering, and replay nonce

Fig. 9. Replay nonce attempt recorded in the experiment log



F. *Experiment Testing*

The experiment results are summarized in Table II. The valid transaction intent was accepted, while all modified or reused transaction intent data was rejected. The results show that the prototype worked as expected in all test scenarios. This means the verification process can detect when the current transaction data no longer matches the user's original signed intent.

TABLE II. RESULT OF TRANSACTION INTENT VERIFICATION

No.	Scenario	Expected Result	Actual Result
1	Valid transaction intent	VALID	VALID
2	Spender address changed after signing	INVALID	INVALID
3	Token address changed after signing	INVALID	INVALID
4	Amount changed after signing	INVALID	INVALID
5	Chain ID changed after signing	INVALID	INVALID
6	Deadline expired	INVALID	INVALID
7	Nonce reused	INVALID	INVALID

G. *Analysis*

The experiment shows that transaction intent verification using EIP-712 typed data signatures can help reduce approval phishing risk. Since the important approval fields are included in the signed data, any change to the spender address, token address, amount, or chain ID causes the verification to fail. This is useful because those fields are commonly targeted in approval phishing scenarios. The system therefore helps users identify when signed data no longer matches the intended approval.

The deadline and nonce fields also improve the security of the verification process. The deadline prevents old signatures from being trusted for too long. The nonce prevents the same signature from being used more than once. These checks are important because EIP-712 defines structured signing, but replay protection still needs to be implemented separately.

However, the prototype still has limitations. It does not send real transactions to the blockchain and does not interact with a real ERC-20 smart contract. The nonce is also stored only in the frontend during runtime, so it resets when the page is refreshed. In a real implementation, nonce validation should be handled by a backend server or smart contract.

Overall, the results show that the proposed prototype can verify transaction intent and detect suspicious changes before a wallet signature is trusted. The system accepted the original signed transaction intent and rejected every modified or reused intent. This supports the use of EIP-712 typed data signatures for clearer and safer approval verification. Therefore, IntentGuard Wallet can be used as a simple experimental model for approval phishing mitigation.

V. CONCLUSION

This study has demonstrated the design and evaluation of IntentGuard Wallet, a prototype for transaction intent verification using EIP-712 typed data signatures. The prototype allows users to connect a MetaMask wallet, create transaction intent data, sign the data, and verify whether the signature still matches the current transaction information. The transaction intent includes important approval fields such as wallet address, spender address, token address, amount, chain ID, nonce, deadline, and verifying contract. The experimental results show that the system can accept valid signed data and reject modified or reused transaction intent data.

The testing results show that spender tampering, token tampering, amount modification, chain ID modification, expired deadline, and replay nonce attempts were all detected as INVALID. This confirms that EIP-712 typed data signatures can help bind a signature to specific structured transaction data. If one important field is changed after signing, the verification process fails because the current data no longer matches the original signed intent. This behavior is useful for mitigating approval phishing, where attackers may try to trick users into approving different data from what they intended.

The nonce and deadline fields also play an important role in improving the verification process. The deadline limits the time period in which a signature can be trusted. The nonce helps prevent replay attempts by rejecting a signature that uses a nonce that has already been recorded. These checks show that EIP-712 is useful for structured signing, but additional application logic is still needed to provide replay protection and stronger approval validation.

However, this prototype still has several limitations. The system does not send real transactions to the blockchain and does not interact directly with an ERC-20 smart contract. The nonce storage is also implemented only in the frontend during runtime, so it resets when the page is refreshed. For future development, nonce validation should be handled by a backend server or smart contract, and the system can be extended to test real token approval transactions in a blockchain testnet environment.

Based on the results, IntentGuard Wallet can be used as a simple experimental model for studying approval phishing mitigation. The system shows that transaction intent verification can make wallet approval safer by checking whether signed data

still matches the user's original intention. Developers can use this approach to design clearer signing flows and reduce the risk of users approving suspicious wallet requests. Future work may include integrating smart contract verification, improving the user interface for non-technical users, and testing the system with real decentralized application approval flows.

SOURCE CODE REPOSITORY AT GITHUB

https://github.com/sarahsurbakti2765/intentguard-wallet

VIDEO LINK AT YOUTUBE

https://youtu.be/uw-zW-Ov-R8

ACKNOWLEDGMENT

The author would like to express sincere gratitude to Dr. Ir. Rinaldi Munir, M.T. for his guidance, lectures, and valuable knowledge throughout the II4021 Cryptography course. The author also appreciates the learning materials, documentation, and development tools that supported the completion of this project. The author hopes that this work can contribute to a better understanding of secure message signing mechanisms and their applications in blockchain systems.

REFERENCES

[1] F. Vogelsteller and V. Buterin, "ERC-20: Token Standard," Ethereum Improvement Proposals, EIP-20, Nov. 2015. [Online]. Available: https://eips.ethereum.org/EIPS/eip-20

[2] R. Bloemen, L. Logvinov, and J. Evans, "EIP-712: Typed structured data hashing and signing," Ethereum Improvement Proposals, EIP-712, Sept. 2017. [Online]. Available: https://eips.ethereum.org/EIPS/eip-712

[3] MetaMask Docs, "Sign data," MetaMask Developer Documentation. [Online]. Available: https://docs.metamask.io/metamask-connect/evm/guides/sign-data/

[4] ethers.org, "Hashing utilities," ethers.js Documentation, version 6. [Online]. Available: https://docs.ethers.org/v6/api/hashing/

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Juni 2026

Sarah Alwa Neguita Surbakti 18223023