

Article

Cryptographic Protocols for Blockchain Systems, Including Protocols for Ensuring the Quantum Stability of Blockchain Systems and Platforms

Evgeniya Ishchukova ^{1,2,*} , Kirill Romanenko ² , Sergei Petrenko ¹, Alexey Petrenko ¹ and Alexey Nekrasov ² 

¹ Scientific Center for Information Technologies and Artificial Intelligence, Sirius University of Science and Technology, Sirius Federal Territory, Sochi 354340, Russia; petrenko.sa@talantiuspeh.ru (S.P.); a.petrenko1999@rambler.ru (A.P.)

² Institute for Computer Technologies and Information Security, Southern Federal University, Chekhova 2, Taganrog 347922, Russia; romanenko.kirill1995@gmail.com (K.R.); alexei-nekrassov@mail.ru (A.N.)

* Correspondence: jekky82@mail.ru; Tel.: +7-(928)-1435898

Abstract

With the development of quantum computing, classical cryptosystems (RSA, ECDSA) that ensure the security of distributed ledgers face an existential threat. This paper examines protocols for protecting personal data (PD) in blockchain, taking into account the “Harvest Now, Decrypt Later” strategy. We propose and formalize a family of protocols designed for storing and exchanging personal data in blockchain systems. The article describes in detail approaches to software implementations of smart contracts for the Ethereum (using ECIES (Elliptic Curve Integrated Encryption Scheme) and Keccak-256) and Hyperledger Fabric 2.5 (integrating NIST post-quantum standards: ML-KEM (Module-Lattice-Based Key Encapsulation Mechanism) and ML-DSA (Module-Lattice-Based Digital Signature Algorithm)) platforms based on the developed protocols. For all developed protocols, a Threat Agent Model (TAM) is presented, threat scenarios are examined, and resilience to typical attack scenarios is demonstrated. A comparative analysis of computational efficiency and overhead is conducted. The results show that using lattice cryptography provides high performance, but the 50-fold increase in signature size makes direct implementation of PQC (Post-Quantum Cryptography) in Layer 1 public networks economically unfeasible. A hybrid model and the use of Layer 2 to ensure quantum resistance are proposed.

Keywords: blockchain; cryptographic protocols; post-quantum cryptography; privacy; Hyperledger Fabric; Ethereum; quantum resistance; NIST PQC standards



Academic Editor: Nik Bessis

Received: 31 May 2026

Revised: 4 July 2026

Accepted: 7 July 2026

Published: 9 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Distributed ledger technologies have transformed approaches to managing digital assets and data [1]. However, the integration of personal data (PD) management into a distributed ledger environment creates a fundamental conflict between blockchain architectural principles and privacy requirements. The inherent properties of blockchain—transaction transparency for network participants and the impossibility of deleting data (immutable ledger)—create critical risks to user privacy. Direct placement of unencrypted PD on a public or even permissioned blockchain violates the “necessary knowledge” principle and makes it impossible to comply with regulatory requirements, in particular the right to be forgotten under the General Data Protection Regulation (GDPR).

Zyskind et al. propose a decentralized personal data management system where users fully control their information, not trusting it to third parties in [2]. The blockchain acts as an automatic access controller and stores only pointers to encrypted data, while the data itself is located in a distributed off-chain repository. Future extensions are also being considered, including confidential computing and node reputation mechanisms. There is a fundamental contradiction between technology and law. The GDPR is built on the idea of centralized responsibility: there is a specific “data controller” who determines the purposes and means of processing, and this controller can be contacted to exercise rights (for example, to delete data). Blockchain, on the contrary, is decentralized, the data in it is often immutable, and responsibility is distributed among many participants.

The most significant problem is legal uncertainty. The GDPR is a technology-neutral and principle-oriented law, which is beneficial for flexibility but detrimental for specificity. The European Data Protection Board (EDPB) could issue official guidelines clarifying how exactly the GDPR’s vague concepts apply to blockchain. This would serve a dual purpose: to help blockchain developers and to bring clarity to the interpretation of the law as a whole [3]. Existing approaches to solving this problem, such as storing hashes instead of the data itself or using off-chain storage, reduce the functionality of systems and deprive them of the integrity guarantee provided by blockchain [4,5].

In this regard, the most promising direction is the implementation of cryptographic methods for protecting data “on the fly” (on-chain encryption) and mechanisms for selective disclosure of information. However, the use of classical encryption schemes to hide data in the blockchain creates a “time capsule” whose security is guaranteed only until the emergence of computing power capable of cracking the cryptographic primitives used. This shifts the problem of storing the PD from the plane of architectural confidentiality to the plane of long-term cryptographic strength. However, their security relies on the difficulty of factorization and discrete logarithm problems. The advent of Cryptographically Relevant Quantum Computers (CRQC) makes the Elliptic Curve Digital Signature Algorithm (ECDSA, secp256k1 curve), used in Bitcoin and Ethereum, vulnerable to private key recovery in polynomial time.

The “Harvest Now, Decrypt Later” (HNDL) strategy poses the greatest risk to personal data: attackers can intercept encrypted data today and decrypt it in the 2030s [6,7]. The immutability of the blockchain means that personal data placed in it today remains accessible to quantum analysis forever [8,9].

Research in other fields also often focuses on the issue of trusted key exchange and data distribution. For example, a fully decentralized authentication and key exchange scheme for the AI-enabled Medical Internet of Things (AIoMT) is proposed in [10]. Unlike classic systems tied to central cloud servers, a decentralized approach to securely share confidential patient medical data is used. The scheme protects personal data and ensures fast generation of session keys without a single point of failure. In vehicular ad hoc networks (VANETs), balancing driver anonymity with traffic safety is critical. A lightweight dual authentication framework deployed on the vehicle side is presented in [11]. The term “conditional anonymity” means that the driver’s identity is hidden from network participants, but trusted law enforcement agencies can disclose it in the event of a crime. The framework is resistant to password attacks and optimized for the limited computing power of on-board vehicle computers. A study presented in [12] focuses on Vehicle-to-Grid (V2G) networks, where electric vehicles can not only charge but also feed excess energy back into the grid. To secure transactions, a multifactor authentication scheme is proposed (combining passwords, biometrics, ID, and physically unclonable functions (PUFs)). The lightweight scheme prevents desynchronization attacks, data forgery, and physical hacking of charging terminals, while ensuring secure encryption key negotiation with minimal

latency. In wearable wireless sensor networks (WBANs), sensors continuously collect a person's medical data. Traditional broadcast encryption requires recipients to know the identities of other group members, leading to data leakage. An “anonymous inner-product decryption” mechanism, allowing each physician to independently decrypt data without knowing the identities of other recipients is introduced in [13]. The technology combines signature and encryption (signcryption), guaranteeing complete anonymity for both sender and recipient.

The subsequent sections of the manuscript are organized as follows: Section 2 provides an overview of the materials and methods. Section 3 outlines the results obtained and their discussion. Section 4 presents the conclusions and recommendations.

2. Materials and Methods

2.1. Statement of the Problem

Common blockchain architectures for protecting personal data are based on a separation principle: the distributed ledger itself stores only hash pointers and access policies, while sensitive information is stored in dedicated off-chain storage—distributed hash tables (DHTs), centralized clouds, or specialized databases. This approach is not a free architectural choice but a necessary compromise due to the fundamental limitations of public blockchain networks.

The first and most obvious limitation is the impossibility of directly storing personal data in the body of transactions or the ledger state. The current European GDPR grants data subjects the right to erasure and also requires minimizing the volume of processed information. The immutable nature of blockchain makes the physical destruction of once-recorded data technically impossible. Furthermore, any information placed in a public ledger is replicated to all full nodes of the network and remains accessible to an indefinite number of people, which directly contradicts the principle of purpose limitation and the requirement of confidentiality. This is why all existing systems are forced to remove actual personal data from the blockchain, limiting themselves to storing only cryptographic links and transaction metadata in the ledger.

The second critical limitation is related to the asymmetric distribution of the load between network participants. In the off-chain encryption and decentralized data storage model within the DHT, the bulk of the computational and communication complexity falls on the end nodes—that is, the users' devices.

At the same time, public network validator nodes, which receive a reward for maintaining consensus, do not bear this cryptographic complexity, as they work exclusively with hash pointers and only verify the formal validity of transactions. This asymmetry is exacerbated as the system scales: each new request for data access or policy change requires the owner to generate a new signature and often re-encrypt the data for the new recipient. For mobile devices with limited computing resources, this burden becomes a critical factor, limiting the practical applicability of such solutions.

Finally, existing off-chain storage schemes create additional vulnerability vectors. Data stored off-chain loses the integrity guarantees provided by the blockchain. Individual DHT nodes can act in a Byzantine manner, replacing or destroying encrypted packets, and replication mechanisms designed to counter this greatly increase storage redundancy and network traffic. Thus, the current tradeoff between privacy protection and architectural integrity comes at the cost of shifting costs to end users and sacrificing the key benefits of a distributed ledger.

These limitations—the impossibility of directly storing personal data on the blockchain, the asymmetric load on end nodes, and the loss of integrity guarantees on the off-chain storage side—form a complex problem that requires a revision of the architectural principles

of privacy protection. The objective of this manuscript is to create a protocol aimed at minimizing the volume of data circulating in plaintext, centralizing cryptographically intensive operations within a formalized smart contract, and ensuring selective access to information.

To achieve this goal, it is necessary to:

1. Develop a protocol for storing and exchanging personal data securely in blockchain systems based on the use of users' public keys.
2. Develop a protocol for validating personal data stored in the blockchain.
3. Develop a protocol for transmitting public keys through the blockchain in a masked form.
4. Develop a protocol for storing and exchanging personal data in a secure manner in blockchain systems based on the use of a shared secret key.
5. Obtain implementations of the developed protocols for public and private blockchains.
6. Obtain experimental data reflecting the load on the blockchain network in terms of data processing speed and stored data volumes.

2.2. Object of Study

The object of the study is a family of protocols that allow for minimizing the amount of open data for various blockchain systems, and it is oriented towards the use of smart contracts as an access control mechanism.

Each user of a blockchain system has a private-public key pair, which is generated upon registration. The user uses their private key to confirm their actions, and any user in the system can verify the accuracy of these records using the user's public key. Blockchain systems strive to avoid storing public keys in plaintext. In public blockchain networks, a user typically shares their public key directly with the user with whom they are interacting. In private blockchains, a dedicated MSP server is responsible for key distribution.

Within the classification of blockchain system users, two main categories are distinguished. The first category includes authorized users, such as operators and owners, who are authorized to view and process personal data. The second category includes validators, whose rights are limited to verifying the presence and authenticity of entries in the registry, but they do not have access to disclosing their actual contents. When developing personal data storage protocols, we will be guided by the following principles:

1. Only authorized users can place data on the blockchain network.
2. Personal data is placed on the blockchain in encrypted form.
3. Any user (both authorized and unauthorized) must be able to verify the validity of the data stored on the blockchain.
4. Only authorized users can retrieve personal data from the blockchain.

The notations that we shall use to describe the protocol are introduced in Table 1.

2.3. Protocol 1: A Protocol for Storing and Exchanging Personal Data Securely in Blockchain Systems Based on the Use of User's Public Keys

The central element of the described system is the message C , consisting of personal data of the user C (DataC) and a unique identifier of the user C (IdC): $C = \text{DataC} + \text{IdC}$.

Consider two system operators (users A and B), both authorized users, who must process the personal data of user C . User C provides user A with their personal data off-chain. User A processes and places user's C data on the blockchain, and user B desires and has a right to receive this personal data for processing. Key pairs in the blockchain system have been generated for users A and B , respectively (pubKa , privKa) and (pubKb , privKb).

A formalized representation of Protocol 1 for storing and exchanging personal data in a secure manner in blockchain systems based on the use of the user's public keys is presented in Table 2.

Table 1. Notations used in the protocol.

Notation	Action
AddressX	User X's address on the blockchain network
IdX	Unique identifier of the user X
DataC	Personal data of the user C
C	Message C of personal data (DataC) and a unique identifier of the user C (IdC)
pubKx	Public key of the user X
privKx	Private key of the user X
p	Parameter of the used elliptic curve (modulus)
kA	Additional key for user A. The numbers kA is random large numbers no greater than the modulus p and are coprime to $(p - 1)$.
kA^{-1}	Additional key for user A. $kA^{-1} \times kA = 1 \bmod (p - 1)$
kB	Additional key for user B. The numbers kB is random large numbers no greater than the modulus p and are coprime to $(p - 1)$.
kB^{-1}	Additional key for user B. $kB^{-1} \times kB = 1 \bmod (p - 1)$
Kab	Shared secret key for users A and B.
$E_{Kab}(M)$	Encrypting message X using a symmetric algorithm on key Kab
$D_{Kab}(M)$	Decrypting message X using a symmetric algorithm on key Kab
$E_{pubKx}(M)$	Encrypt message M with user's X public key
$D_{privKx}(M)$	Decrypt message M with user's X private key
$S_{privKa}(S1, M)$	Obtaining signature S1 for message M using user's X private key
$V_{pubKa}(S1, M)$	Verify signature S1 for message M using user's X public key.
Hash(M)	Hashing message M
Tx	Transaction
t1	Timestamp

Table 2. Protocol 1 for storing and exchanging personal data securely in blockchain systems based on the use of the user's public keys.

No.	From → To	Action
1	A → Blockchain:	$M1 = E_{pubKa}(C)$, $H1 = \text{Hash}(\text{IdC})$, $\text{Tx1} = M1 + H1$ $S_{privKa}(S1, \text{Tx1})$, $H^* = \text{Hash}(C)$
2	B → Blockchain:	IdB, $H2 = \text{Hash}(\text{IdC})$
3	Blockchain → A:	$H1, M1, S1, \text{idB}$ $V_{pubKa}(S1, M1 + H1)$ $C' = D_{privKa}(M1)$
4	A → Blockchain:	$M2 = E_{pubKb}(C')$, $S_{privKa}(S2, M2 + \text{IdB})$ $M1, H1, S1, S2, M2, H^*$
5	Blockchain → B:	$V_{pubKa}(S1, M1 + H1)$, $C = D_{privKb}(M2)$, $V_{pubKa}(S2, C + \text{IdB})$, $H1'' = \text{Hash}(\text{IdC})$, $H^{**} = \text{Hash}(C)$

The Protocol 1 schematic is shown in Figure 1. The process begins with the data recording stage, when user A encrypts a packet with their public key, obtaining a value. Simultaneously, a hash of the identifier is calculated for further retrieval, a transaction signature is generated based on the encrypted block and the hash, and a checksum of the data is calculated. The resulting set of values is entered into the registry.

When access is required, user B submits their ID and the hash of the desired ID to the blockchain. The smart contract automatically checks whether the hashes are equal and ensures that user's B address is on the list of allowed addresses.

Next comes the reencryption and transmission stage. User A restores the original packet by decrypting it with their private key and then reencrypts the data with the recipient's public key. This operation is accompanied by a signature, confirming the data transfer from user A.

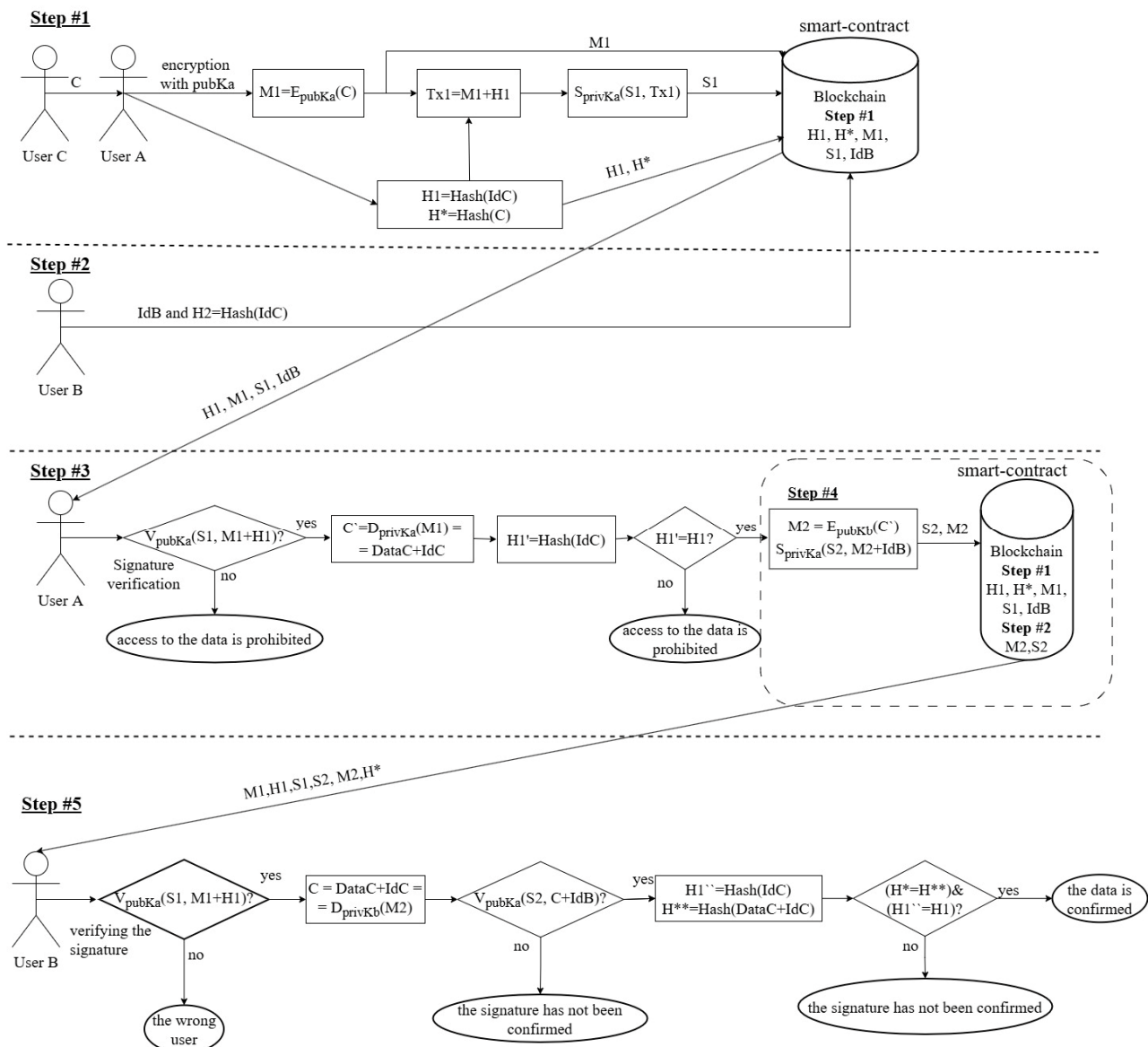


Figure 1. Protocol 1.

The final stage is verification, during which user B retrieves data from the blockchain and decrypts it with their private key. Validation is considered successful if the newly calculated hashes match the reference values recorded in the blockchain.

Protocol 1:

Step 1. User A encrypts message C using the public key $pubKa$. The encrypted message C is stored in M1 ($M1 = E_{pubKa}(C)$). Next, user A creates transaction Tx1, which consists of the encrypted data M1 and the hash H1 of the value IdC ($H1 = Hash(IdC)$, $Tx1 = M1 + H1$). User A then signs transaction Tx1 using a private key $privKa$, creating signature S1 ($S1 = S_{privKa}(S1, Tx1)$). Also, user A creates a hash H^* for message C ($H^* = Hash(C)$). In this step, user A adds the values $\{H1, H^*, M1, S1\}$ to the blockchain.

Note: When implementing the software implementation in a smart contract, it is necessary to set flags depending on which users (based on their IDs) can access the added data (in this example, the data is accessible to user B).

Step 2. User B requests a contract to obtain user's C data. To do so, the user's B identifier (IdB) and the hash of user's C identifier ($H2 = Hash(IdC)$) are transmitted. The contract subsequently verifies that the blockchain contains user's C data ($H2 = H1$) and confirms that user B (IdB) can obtain user's C data (in accordance with the set flags). It is

also subsequently agreed that user's B network address corresponds to user's B public key (to obtain the account's public address, the last 20 bytes of the Keccak-256 hash of the public key are taken, and 0x is appended to the beginning): $IdB = 0x + LSB(Hash(AddressB))$. If the checks fail, the protocol terminates.

Step 3. The contract sends a data set $\{H1, M1, S1, IdB\}$ to user A. User A verifies the data received from the contract using the public key $pubKa$ ($V_{pubKa}(S1, M1 + H1)$). If the verification is successful, user A decrypts the data with the private key ($C' = D_{privKa}(M1) = DataC + IdC$) and checks the hash of the identifier to ensure that the data belongs to user C ($H1' = Hash(IdC)$ and $H1' = H1?$). If the checks fail, the protocol terminates.

Step 4. User A prepares data for user B. To do this, the data is encrypted with user's B public key $pubKb$ ($M2 = E_{pubKb}(C')$) and the transaction is signed with the private key $privKa$ ($S_{privKa}(S2, M2 + IdB)$). User A sends data $M2$ and $S2$ to the contract. The contract verifies that user's A network address matches user's A public key (to obtain the account's public address, the last 20 bytes of the Keccak-256 hash of the public key are taken, and 0x is added to the beginning): $IdA = 0x + LSB(Hash(AddressA))$. If the checks fail, the protocol terminates.

Step 5. The contract transfers a data set $\{M1, H1, S1, S2, M2, H^*\}$ to user B. User B verifies the data signature with user public key $pubKa$ ($V_{pubKa}(S1, M1 + H1)$), and then decrypts the data with his private key $privKb$ ($C = D_{privKb}(M2)$). Next, user B verifies the signature ($V_{pubKa}(S2, C + IdB)$) and checks that the data is indeed from the blockchain registry ($H1'' = Hash(IdC)$ and $H^{**} = Hash(C)$). If the hash verification condition holds ($H^* = H^{**}$) & ($H1'' = H1$), then we can say that the data stored in the blockchain really belongs to user C; user B has received this data and can use it.

2.4. Protocol 2: A Protocol for Validating Personal Data Stored in the Blockchain

Let's consider a situation when an unauthorized user D does not have access rights to the data. Such a user can verify that the data in the registry belongs to the specified user. However, user D will not know the data itself.

There is a system operator (user A), classified as an authorized user, who processes the personal data of user C. User C provides user A with their personal data off-chain. User A processes and places user's C data on the blockchain. User D wants to verify that the data in the registry belongs to user C. Key pairs have been generated for users A and D in the blockchain system: ($pubKa, privKa$) and ($pubKd, privKd$), respectively.

A formalized representation of a protocol for validating personal data stored in the blockchain is presented in Table 3.

Table 3. Protocol 2: A protocol for validating personal data stored in the blockchain.

No.	From → To	Action
1	A → Blockchain:	$M1 = E_{pubKa}(C)$, $H1 = Hash(IdC)$, $Tx1 = M1 + H1$, $H^* = Hash(C)$, $S_{privKa}(S1, Tx1)$
2	B → Blockchain:	IdD , $H2 = Hash(IdC)$, $r1$
3	Blockchain → D:	$H1, M1, S1, IdD, r1$ $V_{pubKa}(S1, M1 + H1)$ $C' = D_{privKa}(M1)$
4	A → Blockchain:	$M3 = E_{pubKd}(IdC + t1)$
5	Blockchain → B:	$M3$

A diagram of Protocol 2 is presented in Figure 2. During the verification process, which does not require direct access to the data content, validator D initiates the exchange by sending a timestamp and hash. User A generates a response message, which is a set of the identifier and the received timestamp, encrypted with the validator's public key $pubKd$. After decryption, validator D compares the transmitted timestamp and the hash

of the identifier. A successful match between these values guarantees that the data in the blockchain truly belongs to entity C and was added to the ledger by the legitimate owner.

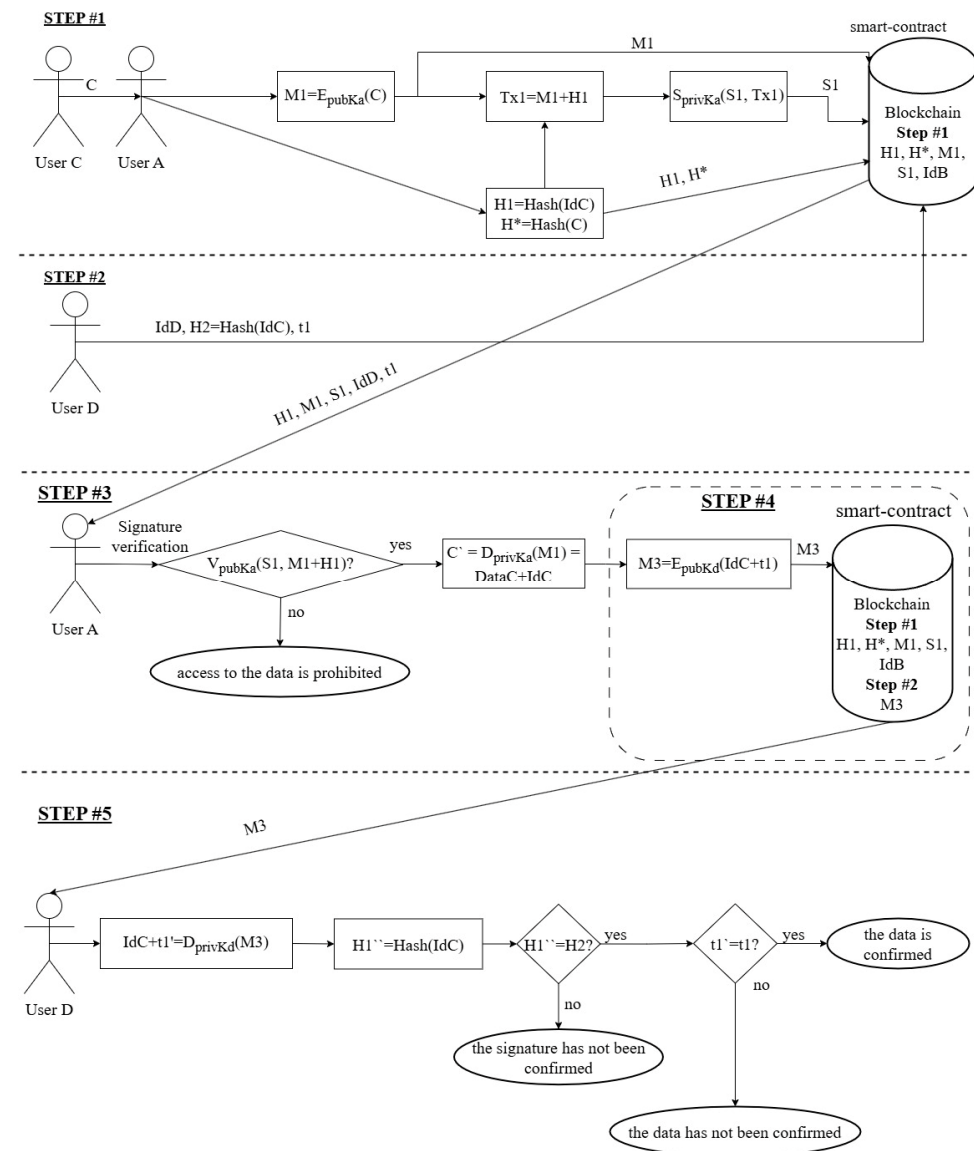


Figure 2. Protocol 2.

Protocol 2:

Step 1. User A encrypts message C using the public key $pubKa$. The resulting encrypted data is written to M1. User A creates transaction Tx1, which consists of the encrypted data M1 and the hash H1 of the value IdC. User A signs transaction Tx1 using the private key $privKa$, creating signature S1. User A also creates a hash H^* of message C. User A transmits the values $H1, H^*, M1$, and $S1$ to the blockchain.

Note: When implementing software in a smart contract, it is necessary to set flags for which users (based on their identifiers) can access the entered data (in the example under consideration, the data is available to user B but not available to user D).

Step 2. User D wants to verify that the ledger stores user's C data and that it is valid. To do this, User D makes a request to the contract. Then, the following parameters are sent: user's D identifier (IdD), the hash of user's C identifier ($H2 = Hash(IdC)$), and a timestamp that will allow the verification of the uniqueness of the transaction

(t1). The contract verifies that user's D blockchain address matches user's D public key: $\text{IdD} = 0x + \text{LSB}(\text{Hash}(\text{AddressD}))$. If the verification fails, the protocol terminates.

Step 3. The contract sends a request to user A (the one who placed the data on the blockchain) to validate user's C data. The request contains the following parameters: the hash of user's C identifier (H1), the encrypted data and user's C identifier (M1), the signed transaction (S1), user's D identifier (IdD), and the timestamp (t1). User A verifies signature S1 using its public key pubKa. If the verification fails, the protocol terminates.

Step 4. User A determines user's C data by recovering it from M1 using a private key privKa ($C = D_{\text{privKa}}(M1) = \text{DataC} + \text{IdC}$). User A extracts user's C identifier (IdC) from M1, adds a timestamp (t1) to it, and encrypts it using user's D public key ($M3 = E_{\text{pubD}}(\text{IdC} + t1)$). User A returns the value M3 to the contract.

Step 5. The contract sends the value M3 to user D. User D extracts user's C identifier and timestamp $t1'$ from M3 using his private key ($M3 = E_{\text{pubKd}}(\text{IdC} + t1')$). User D compares the extracted timestamp $t1'$ with the timestamp t1 of his request in Step 2. User D then verifies that the hash of the extracted user's C identifier matches the hash H2 that he sent to the contract in Step 2. If the checks are successful, user D receives confirmation that the data in the blockchain indeed belongs to user C and was sent to the blockchain by user A.

2.5. Protocol 3: A Protocol for Transmitting Public Keys Through the Blockchain in a Masked Form

In Protocol 1, at Step 4, user A encrypts data using user's B public key. Protocol 1 does not specify how user A obtained user's B public key. The transfer of public keys between participants can occur outside the blockchain, for example, using separate secure communication channels or specialized protocols. We propose considering such a protocol for transmitting public keys through the blockchain in a masked form.

Protocol 3 is based on the mathematical principle of Shamir's three-step encryption, which allows users to securely exchange their public keys over a public blockchain without the use of third-party communication channels. The process resembles the sequential application and removal of locks: first, the sender "masks" their key with a random secret number and transmits it over the blockchain. The recipient then applies their own "mask" and returns the data, after which the sender securely removes their first layer of protection. Thanks to the property of commutativity (the independence of the multiplication order), the recipient only needs to remove their mask using the inverse mathematical algorithm so that eavesdroppers on the blockchain see only encrypted noise, and end users are guaranteed to confidentially obtain each other's clear keys.

The protocol uses a hybrid approach. The blockchain infrastructure is derived from Ethereum's elliptic curve, but the masking algorithm itself operates in a multiplicative residue group modulo $p \in \mathbb{Z}_p^*$, where the base element is the X coordinate of the public key (in compressed format), and the operations are classical exponentiation modulo p. Let's assume that pairs of keys are generated in the blockchain for each user: for user A, the public key pubKa and the private key privKa; for user B, the public key pubKb and the private key privKb.

A formalized representation of the protocol for transmitting public keys through a blockchain in a masked form is presented in Table 4.

A diagram of Protocol 3 is presented in Figure 3.

Protocol 3:

Step 1. Users A and B independently generate additional keys: for user A, a pair of numbers k_A and k_A^{-1} , and for user B, a pair of numbers k_B and k_B^{-1} . The numbers k_A and k_B are random large numbers no greater than the modulus p and are coprime to $(p - 1)$, where the modulus p is the parameter of the used elliptic curve.

Note: These can be found using the extended Euclidean algorithm. The additional keys will be used to transmit public keys securely through the blockchain.

Users A and B raise their public keys to the power of the private key, obtain values X_a and X_b , respectively, and transmit the result to the blockchain.

Step 2. The blockchain returns the value X_a to user B and the value X_b to user A.

Step 3. User B raises the obtained value X_a to the power of his secret k_B . User A raises the obtained value X_b to the power of his secret k_A .

Step 4. The blockchain returns the value Y_a to user B and the value Y_b to user A.

Step 5. User B raises the obtained value Y_a to the power of his secret k_B^{-1} . User A raises the obtained value Y_b to the power of his secret k_A^{-1} .

Step 6. The blockchain returns the value Z_a to user B and the value Z_b to user A. User B raises the resulting value Z_a to the power of their secret, k_B^{-1} , and obtains the public key of user A, pubK_a . User A raises the resulting value Z_b to the power of their secret, k_A^{-1} , and obtains the public key of user B, pubK_b .

Table 4. Protocol 3: A protocol for transmitting public keys through the blockchain in a masked form.

No.	From → To	Action
1	A → Blockchain:	$X_a = (\text{pubK}_a)^{k_A} \bmod p$
	B → Blockchain:	$X_b = (\text{pubK}_b)^{k_B} \bmod p$
2	Blockchain → A:	X_b
	Blockchain → B:	X_a
3	A → Blockchain:	$Y_a = (X_b)^{k_A} \bmod p = (\text{pubB})^{k_B \times k_A} \bmod p$
	B → Blockchain:	$Y_b = (X_a)^{k_B} \bmod p = (\text{pubA})^{k_B \times k_A} \bmod p$
4	Blockchain → A:	Y_b
	Blockchain → B:	Y_a
5	A → Blockchain:	$Z_a = (Y_b)^{k_A^{-1}} \bmod p = (\text{pubA})^{k_B \times k_A \times k_A^{-1}} \bmod p = (\text{pubA})^{k_B}$
	B → Blockchain:	$Z_b = (Y_a)^{k_B^{-1}} \bmod p = (\text{pubB})^{k_B \times k_A \times k_B^{-1}} \bmod p = (\text{pubB})^{k_A}$
6	Blockchain → A:	$Z_b, \text{pubK}_b = (Z_b)^{k_A^{-1}} \bmod p = (\text{pubK}_b)^{k_A \times k_A^{-1}} \bmod p = \text{pubK}_b$
	Blockchain → B:	$Z_a, \text{pubK}_a = (Z_a)^{k_B^{-1}} \bmod p = (\text{pubK}_a)^{k_B \times k_B^{-1}} \bmod p = \text{pubK}_a$

2.6. Protocol 4: A Protocol for Storing and Exchanging Personal Data Securely in Blockchain Systems Based on the Use of a Shared Secret Key

Protocol 1 imposes restrictions on the use of the information system: only the user who processed the personal data can decrypt it and forward it in response to a request. If the user loses access to the system (due to any unforeseen circumstances), all data will be lost. Therefore, it makes sense to implement a modification of Protocol 1 that would store personal data using a symmetric encryption algorithm (AES, Kuznyechik, Magma, and others). In this case, all users with access to the data will be able to process it and respond to a request. Moreover, the output of Protocol 3 can be used to generate a shared secret key by multiple blockchain network users using the Elliptic Curve Diffie–Hellman (ECDH) algorithm.

The initial data for Protocol 4 will be the same as for Protocol 1. User C provides user A with their personal data off-chain. User A processes and places user's C data on the blockchain. User B wants and has a right to receive this personal data for processing. We will assume that users A and B have previously generated a shared secret key, K_{ab} . Moreover, Protocol 4 can be easily extended to any number of users included in the group sharing access to the processed data.

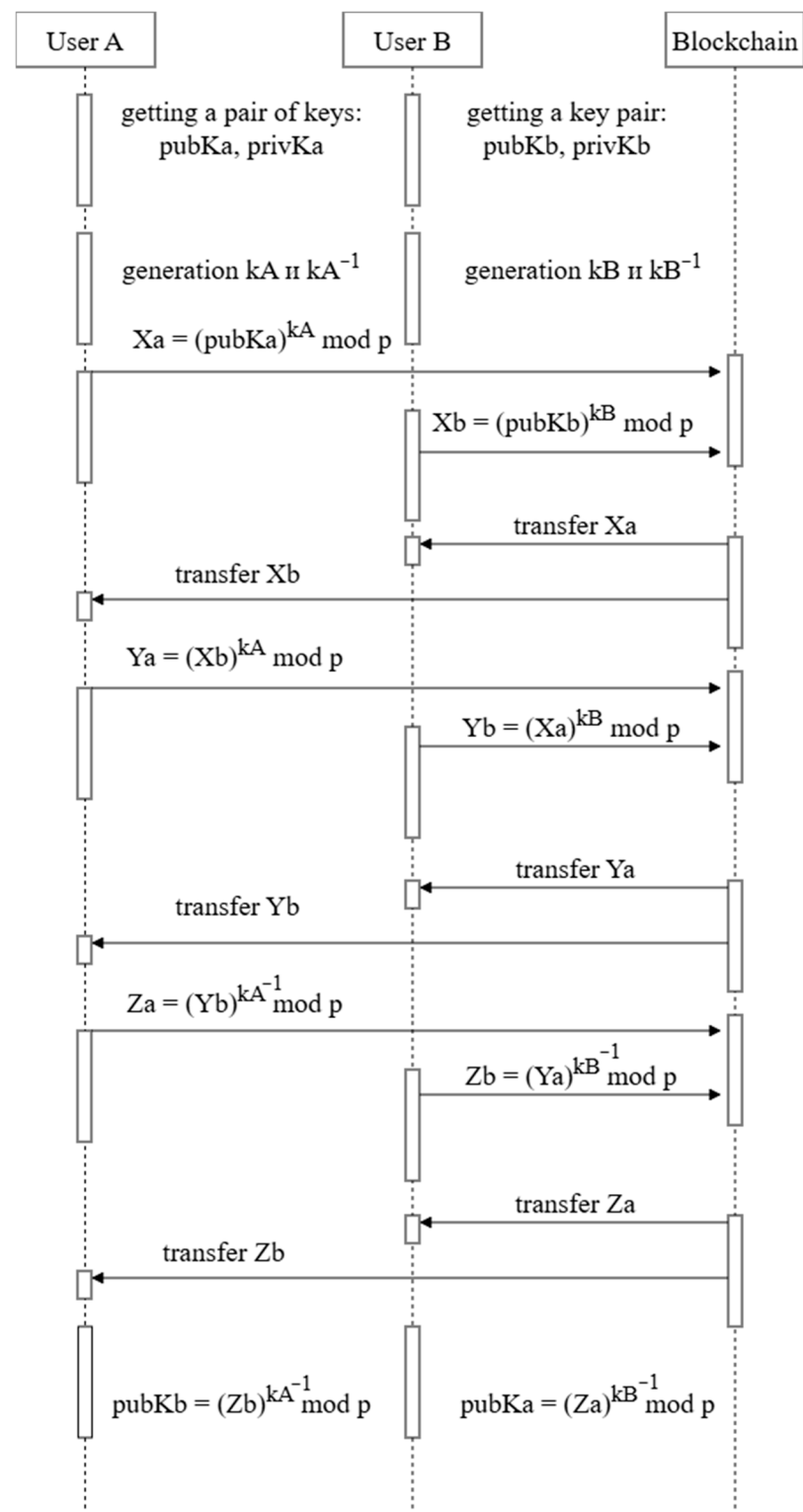


Figure 3. Protocol 3.

A formalized representation of the protocol for storing and exchanging personal data in a secure form in blockchain systems based on the use of a shared secret key is presented in Table 5.

Protocol 4:

Step 1. User A encrypts message C using the shared secret key Kab . He stores the encrypted message C in $M1$ ($M1 = E_{Kab}(C)$). Next, user A generates a hash $H1$ from the

value IdC and a hash H^* from message C . At this step, user A stores the values $H1$, H^* , and $M1$ in the blockchain.

Note: When implementing the software in a smart contract, it is necessary to set flags depending on which users (based on their IDs) can access the added data (in this example, the data accessible to user B).

Step 2. User B requests a contract to obtain user's C data. To do so, he transmit data derived from user's B identifier (IdB) and the hash of user's C identifier ($H2 = Hash(IdC)$). The contract subsequently verifies that the blockchain contains user's C data ($H2 = H1$) and confirms that user B (IdB) can obtain user's C data (in accordance with the set flags). It is also subsequently agreed that user's B network address corresponds to user's B public key (to obtain the account's public address, the last 20 bytes of the Keccak-256 hash of the public key are taken, and $0x$ is appended to the beginning): $IdB = 0x + LSB(Hash(AddressB))$. If the checks are not passed, the protocol terminates.

Step 3. The contract sends a data set $\{M1, H1, H^*\}$ to user B . User B decrypts the data with the shared secret key K_{ab} ($C = D_{K_{ab}}(M1)$) and retrieves user's C data ($C = DataC + IdC$). User B verifies that the data is indeed from the blockchain ledger ($H1 = Hash(IdC)$ and $H^* = Hash(C)$). If the checks are completed, then we can say that the data stored on the blockchain truly belongs to user C ; user B has received this data and can use it.

The diagram of Protocol 4 is shown in Figure 4. It can be seen that Protocol 4 has become less loaded compared to Protocol 1.

Table 5. Protocol 4: A protocol for storing and exchanging personal data in a secure form in blockchain systems based on the use of a shared secret key.

No.	From → To	Action
1	$A \rightarrow \text{Blockchain}$:	$M1 = E_{K_{ab}}(C)$, $H1 = Hash(IdC)$, $H^* = Hash(C)$, $Tx1 = M1 + H1$
2	$B \rightarrow \text{Blockchain}$:	IdB , $H2 = Hash(IdC)$
3	$\text{Blockchain} \rightarrow B$:	$M1, H1, H^* C = D_{K_{ab}}(M1)$

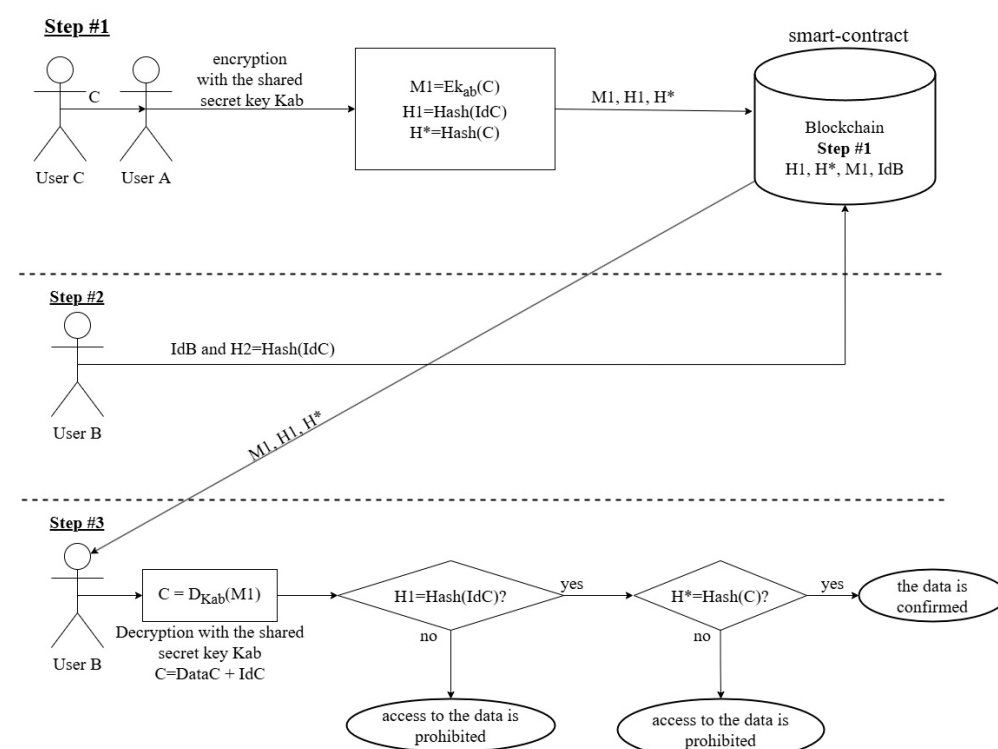


Figure 4. Protocol 4.

2.7. Defining the Threat Agent Model

Let's introduce assumptions and definitions for constructing a TAM. The assumptions define the properties of the environment, infrastructure, and algorithms that the protocol developer considers to be guaranteed to be secure. We introduce the following assumptions for the four developed protocols.

Strength of cryptographic primitives. Asymmetric encryption algorithms (E_{pubK} , D_{privK}) and electronic signatures are computationally intractable to brute-force attacks. Knowledge of the public key or ciphertext does not allow recovery of the private key. We assume that asymmetric encryption is IND-CCA2-strong (Indistinguishability under Adaptive Chosen-Ciphertext Attack). We assume that the electronic signature algorithm used has the EUF-CMA (Existential Unforgeability under Chosen Message Attacks) property. Hash functions (Hash, Keccak-256) are resistant to first/second primitive searches and resistant to collisions. A symmetric encryption algorithm (e.g., AES-GCM or Kuznyechik) is completely secure. An attacker cannot recover the plaintext M_1 from the ciphertext C without knowledge of the key K_{ab} by brute force or cryptanalysis.

Smart contract reliability. Smart contract logic (checking access flags, validating the sender's address against its public key) executes correctly and is not susceptible to reentrancy vulnerabilities or external manipulation of state variables.

Blockchain integrity and persistence. The smart contract is executed in a trusted distributed ledger environment. Data recorded in the blockchain cannot be modified or deleted by an attacker (transaction rollback protection).

Security of legitimate users' private keys. Users keep their private keys secret. Compromising a private key means a complete hack of the participant and is beyond the scope of the security analysis of the protocol itself.

Trust in user's A data. It is assumed that user A initially possesses the legitimate data of user C ($\text{DataC} + \text{IdC}$) and has the right to manage it.

Honesty and Availability of the Data Creator (User A). User A is assumed to be online and willing to honestly respond to contract queries. If A is offline or sabotages the process, the protocol legitimately fails.

Timestamp Uniqueness (t1). The local clocks of user D and the blockchain are synchronized with sufficient accuracy to ensure transaction uniqueness and prevent time collisions.

Extended Euclidean Security. For any random integer $k \in [1, p - 1]$, there always exists a uniquely calculable inverse $k^{-1} \bmod (p - 1)$ such that $k \times k^{-1} \equiv 1 \bmod (p - 1)$.

Perfect Random Number Generator (PRNG). Users A and B generate their secret multipliers k_A and k_B independently of each other using a cryptographically secure PRNG. An adversary cannot predict or narrow the search space for these numbers.

Integrity of the Transit Environment (Blockchain). The blockchain ensures that intermediate values are delivered to their destinations without modification during the transit process between steps.

Let's introduce the TAM. The model is based on the Dolev-Yao concept, adapted to the specifics of blockchain networks. Attacker E can be either external (an attacker within the network) or internal (a malicious blockchain user without access rights to specific data). The attacker has the following capabilities.

Passive Observer (Global Interceptor). Has full access to the blockchain ledger. Sees and records absolutely all intermediate values.

Active Attacker (MitM/Address Space Attacker). Can attempt to send his transactions to the blockchain, generate address collisions, or impersonate participants in an attempt to interfere with the exchange process (passing off his own intermediate value as the value of any user).

Transaction Generation. Can send his own to the smart contract with any arbitrary arguments on behalf of any address he controls.

Address space attack. May attempt to find a public key for which the last 20 bytes of the Keccak-256 hash match the address of a legitimate user A or B (collision search attack/vanity address search).

2.8. Security Proof for Protocol 1

Let's identify the attacker's goals (attack vectors that the protocol must mitigate).

Attack 1: Unauthorized Access to Data (Confidentiality Breach)

Scenario: Attacker E (who is not user B and does not have an access flag in the smart contract) attempts to obtain user's C data.

Method: E initiates Step 2 by sending a request to the contract. The goal is to bypass the flag check, spoof IdB, or force the contract to send it the dataset from Step 3. Even if the contract rejects the request, E attempts to intercept the dataset {H1, M1, S1, IdB} in Step 3 or {M1, H1, S1, S2, M2, H*} in Step 5 in the communication channel and decrypt M1 or M2.

Attack 2: Identity Spoofing/ Address Space Manipulation (MitM)

Scenario: Attacker E attempts to impersonate user B in a contract and user A in order to redirect data to their own address.

Method: E generates a key pair whose address matches user's B address (or exploits the address-to-public key binding vulnerability in Step 2) to pass the network address-to-account public address match check.

Attack 3: Data Substitution or Modification (Integrity Attack)

Scenario: An attacker attempts to replace encrypted data M1 or M2 with corrupted or deliberately false data to mislead user B.

Method: Intercept messages in Steps 3 or 4 and modify the transaction contents. The goal is to force the hash check in Step 3 ($Tx1' = M1 + H1$) or Step 5 ($(H^* = H^{**}) \ \& \ (H1'' = H1)$) to converge with invalid data.

Attack 4: Replay Attack

Scenario: The attacker intercepts valid transactions/messages from a previous exchange session and resends them.

Method: Resending the (M2, S2) pair to the contract in Step 4 to falsely indicate that the actual data was transmitted.

To construct a rigorous academic proof of the security of Protocol 1, we use the provable security method. We formally reduce the protocol's security to the strength of its underlying cryptographic primitives (asymmetric encryption and hash functions) using game-theoretic models (Games/ Adversary).

To prove resistance to Attack 1, we consider two threat scenarios (Theorems 1 and 2).

Theorem 1. *Protection against smart contract flag bypass: An attacker E cannot forge IdB or bypass flag verification to obtain a dataset in place of the legitimate user B.*

Proof. Authorization mechanism in Step 2: According to the protocol description, the smart contract fulfills a mandatory cryptographic condition: it verifies that the blockchain address of the transaction sender strictly corresponds to his public key.

On-chain authentication: In a blockchain architecture (e.g., Ethereum), the transaction sender's address (msg.sender) is calculated automatically at the EVM level by recovering the public key from the signature on the transaction itself (ECDSA). The contract verifies: $\text{AddressE} = 0x + \text{LSB}_{20}(\text{Keccak-256}(\text{pubKe}))$.

Logical barrier: An attacker E can send any fake identifier IdB in the transaction body. However, the smart contract matches this IdB with the real msg.sender(AddressE) from which the transaction originated.

Blocking: Since the access flags in the smart contract in Step 1 were set only for the legitimate user's B address (AddressB), and the transaction originated from AddressE, the verification ($\text{msg.sender} = \text{AddressB}$) fails.

Conclusion: The contract terminates. The Step 3 generation scenario addressed to the intruder is blocked at the blockchain consensus level. $\square$

Theorem 2. *Protocol 1 is secure against interception in communication channels: If an attacker E acts as an active eavesdropper in a network channel and can intercept the legitimate dataset $\{H1, M1, S1, IdB\}$ in Step 3 or the full dataset $\{M1, H1, S1, S2, M2, H^*\}$ in Step 5, he will not be able to decrypt M1 or M2 to obtain DataC.*

Proof. Analysis of the cryptographic state of the intercepted data: suppose attacker E intercepts absolutely all protocol messages. He has access to the following data: $H1, H^*, Id_B$, and two encrypted blocks:

$M1 = E_{\text{pubKa}}(\text{DataC} + IdC)$ – encrypted with user's A public key.

$M2 = E_{\text{pubKb}}(\text{DataC} + IdC)$ – encrypted with user's B public key.

Formal Reduction: Suppose an attacker E possesses the Crack(M) algorithm, which allows him to extract DataC from ciphertexts M1 or M2 without knowing the private keys privKa or privKb . Let's construct a simulator Sim that uses this algorithm to crack the classical encryption scheme in the standard cryptographic game IND-CCA2.

- ✓ The IND-CCA2 Challenger generates a key pair ($\text{pubK}^*, \text{privK}^*$) and sends the public key (pubK^*) to the simulator Sim.
- ✓ The simulator Sim initiates Protocol 1 and substitutes the resulting pubK^* for user's B public key.
- ✓ The Challenger selects a random bit $b \in \{0, 1\}$, encrypts message C_b with key pubK^* and returns the target ciphertext M^* to the simulator Sim.
- ✓ Simulator Sim transmits this ciphertext M^* to attacker E under the guise of the intercepted message M2 in Step 5.
- ✓ Attacker E runs its Crack(M^*) algorithm, successfully extracts the plaintext message C_b , and returns it to the simulator.
- ✓ Simulator Sim instantly guesses bit b and wins the IND-CCA2 game with probability 1.

Contradiction: By assumption, the asymmetric encryption scheme used in the protocol is strictly IND-CCA2-secure, meaning it is impossible for Sim to exist with a winning probability other than random guessing ($1/2 + \epsilon$).

Conclusion: It is mathematically impossible for attacker E to have Crack(). $\square$

General conclusion regarding Attack 1: User's C privacy is completely protected. Attacker E cannot access the data either by manipulating the smart contract logic (due to strict on-chain authentication of the sender's address) or by passively intercepting traffic (due to the proven strength of IND-CCA2 asymmetric encryption). The protocol is resistant to Attack 1.

To prove resistance to Attack 2, we prove Theorems 3 and 4.

Theorem 3. *Protocol 1 is resistant to identity (address space) spoofing: Attacker E cannot hijack user's B access rights by misrepresenting his own public key as user's B legitimate address in the smart contract.*

Proof. Address Mathematics: Steps 2 and 4 of the protocol rigidly define the rule for generating an account's public address on the blockchain: $\text{AddressX} = 0x + \text{LSB}_{20}(\text{Keccak-256}(\text{pubKx}))$.

Attempt to Attack: To impersonate user B, attacker E must send a request to the smart contract from his account, but trick the contract into believing that his public key corresponds to user's B authorized address (pass access flag validation).

Reduction to a cryptographic problem: The smart contract reads the transaction sender's address (msg.sender), which is mathematically derived from the user's public key. Attacker E has his key pubKe and his address AddressE. To pass the flag check set for IdB, the collision condition must be met: $\text{LSB}_{20}(\text{Keccak-256}(\text{pubKe})) = \text{LSB}_{20}(\text{Keccak-256}(\text{pubKb}))$.

Computational complexity: The address length is 20 bytes (160 bits). Finding a collision for the 160-bit Keccak-256 hash function requires, on average, 2^{80} operations (due to the birthday paradox). A targeted impersonation attack (finding the first primitive for a specific address of user B) requires 2^{160} operations.

Conclusion: This computational problem is unsolvable in polynomial time. Attacker E is unable to forge the address binding and will be blocked by the smart contract at Step 2. $\square$

Theorem 4. *Protocol 1 is resistant to the MitM attack: An attacker E will not be able to obtain user's C data (Data C) even if he were able to intercept off-chain traffic control and trick user A into believing in Step 4 that user's E address is user's B address by sending data to a contract for E.*

Proof. Cryptographic action in Step 4: When user A prepares data to send (whether to the legitimate B or the disguised E), they perform a rigorous mathematical operation: $M2 = E_{\text{pubKb}}(C')$. The data is encrypted with the recipient's public key.

Branch logic (Attacker's dilemma):

- ✓ Case A (If A uses the public key pubKb): Even if E intercepts this packet M2 in the communication channel or from the smart contract in Step 5, he will not be able to read it. To decrypt M2, he needs the private key privKb, which only user B possesses. The attacker will only receive encrypted data, which makes the MitM attack pointless.
- ✓ Case B (If E slipped user A his public key pubKe): If in Step 4 user A encrypts data with the attacker's key ($M2 = E_{\text{pubKe}}(C')$), E will be able to decrypt it in Step 5. However, the attack will still fail at the final verification stage. In Step 5, the recipient is required to verify the signature S2 created by user A: $V_{\text{pubKa}}(S2, M2 + \text{IdB})$. User A signs a hash containing the strong identifier of the legitimate user B (IdB). In Step 5, according to the protocol, the recipient must verify the signature $V_{\text{pubKa}}(S2, \text{DataC} + \text{IdC} + \text{IdB})$. Since the attacker E has an identifier $\text{IdE} \neq \text{IdB}$, the signature equality check will fail. The protocol will immediately terminate, and the attacker E will not be able to legitimately use or verify this data in the system.

Conclusion: $\square$

General conclusion regarding Attack 2: Protocol 1 is fully protected against identity spoofing and man-in-the-middle (MitM) attacks. Attacker E is unable to mathematically compromise the blockchain's address space, and any attempt to spoof off-chain parameters results in the destruction of cryptographic signature and hash checks at Step 5, preventing unauthorized use of the data.

To prove the resistance of Protocol 1 to Attack 3 (Data Forgery or Modification/Integrity Violation), we mathematically justify why an attacker E cannot undetectedly modify encrypted data M1 or M2. The proof is based on two independent defenses: the cryptographic strength of user's A digital signature and the strong collision resistance of hash functions.

Theorem 5. *In Protocol 1, data M1 is protected from forgery: Attacker E will not be able to forge data M1 under any circumstances.*

Proof. Threat: Attacker E intercepts the off-chain dataset $\{H1, M1, S1, IdB\}$ sent by the smart contract in Step 3 and replaces the encrypted message $M1$ with a falsified message $M1^*$, attempting to misinform user A or disrupt the decryption logic. Its goal is to ensure that the signature $V_{pubKa}(S1, M1^* + H1)$ is verified.

Mathematical relationship of the parameters: In Step 1, user A generated signature $S1$ for transaction $Tx1 = M1 + H1$ using his private key: $S_{privKa}(S1, M1 + H1)$.

Formulation of the attacker's problem: To replace the data in $M1^*$ and successfully pass validation in Step 3, attacker E must solve one of two equations:

- ✓ Option A (Using the old signature): Find $M1^* \neq M1$ for which the existing signature $S1$ remains valid: $V_{pubKa}(S1, M1^* + H1) = \text{true}$. This is possible only if the condition $M1^* = M1$ is met. Any change to even one bit of the data will violate the identity.
- ✓ Option B (Generating a new signature): Calculate a counterfeit signature $S1^*$ corresponding to the modified data: $S_{pubKa}(S1^*, M1^* + H1)$.

Reduction to the cryptographic problem: By assumption, the digital signature algorithm used has the EUF-CMA property. This means that an attacker E, who does not possess the private key $privKa$, has a negligible (close to zero) probability p of successfully forging the digital signature for the new message.

Conclusion: Attacker E cannot bypass verification. Any modification of $M1$ in Step 3 will cause verification to fail, and the protocol will terminate immediately. $\square$

Theorem 6. In Protocol 1, data $M2$ is protected from tampering: Attacker E cannot tamper with data $M2$ under any circumstances.

Proof. Threat: Attacker E intercepts the transaction at Step 4 and replaces the encrypted data $M2$ with false data $M2^*$. Their goal is to trick user B at Step 5 into accepting the falsified data ($DataC^*$) and bypass the end-to-end conjunction of the verification hashes ($(H^* = H^{**}) \ \& \ (H1' = H1)$).

Mathematical relationship of parameters: In Step 1, the reference hash of the original message was recorded in the immutable blockchain ledger: $H^* = \text{Hash}(C) = \text{Hash}(DataC + IdC)$

Verification logic in Step 5: Upon receiving the packet, user B decrypts it with his private key: $(DataC^* + IdC^* = D_{privKb}(M2^*))$. It then calculates a verification hash from the obtained result: $(H^{**} = \text{Hash}(DataC^* + IdC^*))$. The protocol succeeds only if the calculated H^{**} is exactly equal to the reference H^* from the blockchain.

Reduction to the search for a collision of the second kind: For the substitution of data in $M2^*$ to benefit the attacker E, the decrypted text ($DataC^* + IdC^*$) must contain deliberately false information desired by the attacker but still satisfy the condition: $\text{Hash}(DataC^* + IdC^*) = H^*$, where $H^* = \text{Hash}(DataC + IdC)$ is the hash of the legitimate data. This is the classic problem of finding the second preimage (Second Preimage Resistance).

Computational complexity: By assumption, the hash function used (e.g., cryptographically strong SHA-256 or Keccak-256) has absolute resistance to the search for the second preimage. For a 256-bit function, the computational complexity of finding such a collision is 2^{256} operations. This problem is unsolvable in polynomial time.

Additional barrier (Verifying signature $S2$): Even if attacker E somehow bypassed the hash, he would also need to forge the signature $S_{privKa}(S2, M2 + IdB)$. In Step 5, user B verifies the signature $V_{pubKa}(S2, DataC^* + IdC^* + IdB)$. Altering the data inside $M2^*$ would immediately break this equality, since the attacker does not have the key $privKa$ to sign the modified packet.

Conclusion: $\square$

General conclusion regarding Attack 3: Protocol 1 is completely resistant to any attempts to spoof or modify transit data. Due to the immutable recording of reference hashes in the blockchain at Step 1 and strict end-to-end verification of user's A signatures, any change to even a single bit in the encrypted data M1 or M2 makes it mathematically impossible to successfully pass the checks at Steps 3 and 5. The protocol guarantees the integrity of the transmitted information.

To prove the resilience of Protocol 1 to Attack 4 (Replay Attack), we must analyze the consequences of attacker E resending a previously intercepted valid data pair (M2, S2) to the smart contract in Step 4. The proof is based on the logic of the smart contract's on-chain states and the cryptographic consistency of the session parameters.

Theorem 7. *Protocol 1 is resistant to replay attacks: A replay attack is guaranteed to be thwarted at one of two levels of the system: either at the level of blockchain and smart contract security (on-chain) or at the level of end-to-end data verification by the recipient (off-chain).*

Proof. Threat: Assume that attacker E passively intercepts valid encrypted data M2 and signature S2, which user A sent to the contract in Step 4 during a previous (previous) successful exchange session. Later, within a new session (or outside of it), E rebroadcasts this pair (M2, S2) to the smart contract. The attacker's goal is to falsely record in the system the fact of a new transmission of current data to user B and force them to accept outdated information.

On-Chain Blockchain Replay Protection (Transaction Layer). If attacker E attempts to send the pair (M2, S2) to the smart contract by executing a function call on his own behalf, the contract in Step 4 strictly enforces the following condition: the sender's network address matches the public key of the data author (user A). Since the transaction is signed with attacker's E key, the address of the transaction author will be equal to AddressE. The check for a match with user's A address (AddressA) will fail. The contract will reject the transaction.

If E attempts to send an exact copy of user's A old transaction (including user's A original transaction signature) to the network, the basic mechanisms of blockchain (the ideal distributed ledger model) come into play: in networks like Ethereum, each transaction contains a strictly consistent counter—a Nonce (for accounts) or a unique transaction identifier. The blockchain will block a duplicate transaction with a previously used Nonce at the network consensus level. Rewriting an old transaction to the blockchain is physically impossible.

Data Concatenation and Uniqueness of Session Hashes (Business Logic Layer). Suppose an attacker is acting within the current session, where user B has just made a new request in Step 2, the contract has returned new data in Step 3, but the attacker has hijacked the Step 4 channel and, instead of user's A fresh response, has inserted an old packet (M2', S2') from the previous session. In this case, the attack will completely fail in Step 5 when user B verifies:

- ✓ Reconciliation with the ledger: In Step 5, user B must verify: $((H^* = H^{**}) \& (H1'' = H1))$, where H^* and $H1$ are the reference hashes extracted from the current transaction Tx1, which was added to the blockchain in Step 1 of the current session.
- ✓ If the old intercepted packet (M2', S2') contained different data from user C, then after decryption, the calculated hash $H^{**} = \text{Hash}(\text{DataC}' + \text{IdC}')$ will not match the latest reference H^* from the blockchain ($H^* \neq H^{**}$).
- ✓ If user's C data has not changed, but the attacker simply resends the message, user B verifies the signature $V_{\text{pubKa}}(S2', \text{DataC} + \text{IdC} + \text{IdB})$ in Step 5. Since the signature S2' is mathematically hard-wired to the old ciphertext M2' via the hash function $S_{\text{privKa}}(S2,$

M2 + IdB), any mismatch between the context of the current step and the old ciphertext (e.g., if the internal format or metadata has changed) will result in a validation error.

Conclusion: $\square$

General conclusion regarding Attack 4: Protocol 1 is resistant to replay attacks. The basic properties of the blockchain infrastructure prevent transaction replays at the network level, and the rigid mathematical binding of off-chain packets in Step 5 to the unique on-chain hashes of the current session H^* makes sending stale data useless. An attacker E cannot misinform participants. The protocol is secure.

We summarize all the protective mechanisms of Protocol 1 in Table 6.

Table 6. Safeguards of Protocol 1.

Point of Attack	Protocol Protective Barrier	Result for the Offender
Blocking Matrix for Attack 1		
Attempt to forge IdB in a contract	Validating an address on the blockchain network at the EVM level	Transaction is rejected, Step 3 is not activated.
Intercepting packet M1 at Step 3	Encryption using the pubKa key	The packet is useless: to decrypt it, you need the secret privKa, which E does not have.
Intercepting packet M2 at Step 5	Encryption using the pubKb key	The packet is useless: to decrypt it, you need the secret privKb, which E does not have.
Blocking Matrix for Attack 2		
Attempt to guess a public key for someone else's address	Keccak-256's resistance to 2^{160} operations of preimage search	The attacker is cut off by the smart contract in Step 2.
Mitigation of Communications Channels (MitM)	Encrypting the M2 packet with the pubKb key	The packet cannot be decrypted without user's B private key.
Attempt to replace IdB with your own IdE inside the signature	Validating the S2 signature on the pubKa key in Step 5	The check of the conjunction of hashes and identifiers fails, and the protocol terminates abnormally.
Blocking Matrix for Attack 3		
Modification of M1 at Step 3	EUF-CMA signature strength S1 on the pubKa key	Signature S1 fails verification. The protocol terminates.
Modification of M2 at Step 4	The resistance of a hash function to searching for the second preimage is 2^{256} operations.	The hash avalanche effect results in $H^* \neq H^{**}$. The data is rejected. The protocol terminates.
Attempt to forge a signature using altered data	Verification of signature S2 in Step 5 by user B	User's A signature mismatch. Protocol terminates.
Blocking Matrix for Attack 4		
Re-calling a contract function on behalf of E with old data (M2', S2')	On-chain verification of the transaction author's address in Step 4	The transaction is rejected by the smart contract.
Attempt to repeat user's A entire transaction	Nonce mechanism/replay protection at the blockchain consensus level	The network rejects the transaction as a duplicate
Substitution of a fresh response for an old one (M2', S2') in an off-chain channel	End-to-end verification of hash conjunction ($H^* == H^{**}$) in Step 5	The computed hash of the old data does not match the current on-chain hash of the current session.

2.9. Security Proof for Protocol 2

Let's consider the main goals and attack vectors of an adversary for Protocol 2.

Attack 1: Unauthorized Data Disclosure (Privacy Breach). The adversary (or the verifier D) attempts to extract the identifier IdC or the message body C from the intercepted parameters (M1, M3, and H1).

Attack 2: Verification Falsification (Audit Bypass). The adversary attempts to generate a fake response M3' for user D to convince them that valid data for user C is on the blockchain, when in fact it is not there or belongs to someone else.

Attack 3: Replay Attack. The adversary intercepts the old valid response M3 from Step 4 and attempts to send it to user D in a new verification session to disguise the fact that the actual data C has been deleted or modified.

To prove the security of Protocol 2 against Attack 1 (Unauthorized Data Disclosure/Privacy Violation), we prove that neither an external adversary E nor the verifying user D (who has limited rights) can compute the identifier IdC or the message content C from the parameters circulating in the network. The proof is divided into three layers, corresponding to three data elements: M1, H1, and M3.

Theorem 8. *Protocol 2 is resistant to an attack aimed at unauthorized data disclosure: an unauthorized user cannot compute the identifier IdC or the message content C from the parameters circulating in the network.*

Proof. Level 1. Unable to extract data from array M1

Threat: An attacker E or verifier D intercepts data M1 from the blockchain or an off-chain transit channel in Step 3 and attempts to decrypt it to extract the message C and the identifier IdC.

Cryptographic status of M1: According to Step 1 of the protocol, user A encrypted the message C using the public key: $M1 = E_{pubKa}(C) = E_{pubKa}(DataC + IdC)$

Formal reduction: Suppose that an attacker E or verifier D possesses the mathematical algorithm Crack(M1), which is capable of cracking this ciphertext. Let's construct a simulator Sim that uses this algorithm to win the standard cryptographic game IND-CCA2 (indistinguishability of ciphertexts) against Challenger.

- ✓ Challenger generates a key pair (pubK*, privK*) and sends (pubK*) to Sim.
- ✓ The simulator Sim declares this key to be user's A public key.
- ✓ The Challenger encrypts one of two secret messages, C0 or C1, with this key and returns the target ciphertext, M*, to the simulator.
- ✓ The simulator sends M* to the attacker as the intercepted M1. The attacker runs the Crack(M*) algorithm and returns the decrypted message to the simulator. The simulator wins the IND-CCA2 game with probability $p = 1$.

Conclusion: Since the encryption scheme used is assumed to be completely secure, the probability of winning in IND-CCA2 cannot exceed random guessing. Therefore, extracting data from M1 without knowing the private key privKa, is mathematically impossible.

Level 2. Failure to Retrieve IdC from Hash H1

Threat: Attacker E attempts to recover the real identifier IdC, knowing its public hash H1, published on the blockchain.

Mathematical Status of H1: In Step 1, the equality $H1 = \text{Hash}(\text{IdC})$ was established.

Irreversibility Property: Under the assumption introduced, the hash function used has strong first-preimage resistance. This means that for a given value of H1, calculating an argument X such that $\text{Hash}(X) = H1$ requires a brute-force search over the entire space of possible identifiers.

Conclusion: If the space of IdC identifiers has sufficient entropy (e.g., a random 256-bit string), the computational complexity of the attack is 2^{256} operations, which is not feasible in polynomial time. An attacker cannot recover IdC from the H1 hash.

Level 3. Protecting Privacy in Off-Chain Response M3

Threat: In Step 5, user D receives M3. An external attacker E, attempts to intercept it in the communication channel and obtain the IdC. Meanwhile, user D successfully decrypts M3 and extracts the IdC. We will prove that this does not violate data privacy.

Case A: Protection from External Adversary E.

In Step 4, user A generates a response: $M3 = E_{\text{pubKd}}(\text{IdC} + t1)$. The data is encrypted with verifier's D public key. Accordingly, external attacker E, who does not have the private key privKd , faces the same IND-CCA2 complexity of the asymmetric cipher as at Level 1. For E, the data M3 remains unreadable noise.

Case B: Isolating Data from Verifier D.

In Step 5, user D decrypts M3 and indeed extracts the IdC. However, this does not violate privacy. According to Step 2, to even start the protocol, user D must already know the hash of the user they are searching for: $H2 = \text{Hash}(\text{IdC})$. Protocol 2 is designed not to search for unknown users, but to audit the presence of a known user. By extracting IdC, user D merely checks that it matches their previously sent request: $\text{Hash}(\text{IdC}) == H2$. Crucially, the body of message C (DataC) is not used at all in Steps 4 and 5. User A extracts only the identifier from M1, adds a timestamp, and sends it back. The body of message C remains locked within M1 on user's A key.

Conclusion: $\square$

General conclusion regarding the resilience of Protocol 2 to Attack 1: The protocol is completely resistant to unauthorized data disclosure. The confidentiality of the secret message C is protected by user's A end-to-end asymmetric encryption, and the architecture of the interactive response in Step 4 ensures that verifier D receives only the technical verification token ($\text{IdC} + t1$), preserving the secrecy of message C itself.

To prove the resilience of Protocol 2 to Attack 2 (Verification Forgery / Audit Bypass), we must mathematically and logically justify why an attacker E cannot generate a false message $M3'$ that would deceive user D into believing the audit has been successfully passed, even if the data in the blockchain is missing or compromised. The proof is divided into two related parts: the impossibility of forging the cryptographic context and the impossibility of passing the end-to-end checks in Step 5.

Theorem 9. *Protocol 2 is resistant to the Verification Falsification/Audit Bypass attack: an unauthorized user cannot generate a false $M3'$ message that would deceive user D into believing the audit has been successfully completed if the data is missing or compromised in the blockchain.*

Proof. Part 1. Cryptographic Impossibility of Synthesizing an $M3'$ Packet

Threat: Attacker E intercepts user's D request in Step 3 or simply imitates user's A response in Step 4 by sending fake $M3'$ data encrypted with user's D public key pubKd to the contract. The goal is to deceive user D in Step 5 into believing that this response came from the legitimate user A.

Mathematical structure of a legitimate response: In Step 4, user A must extract IdC from the original on-chain message M1 and form a packet: $M3 = E_{\text{pubKd}}(\text{IdC} + t1)$.

Attacker's E constraints: To generate a successful forgery $M3'$, the attacker must embed the correct IdC within the ciphertext, corresponding to the verified hash H2.

Logical contradiction: If user's C data does not exist in the blockchain at all, then the real IdC is unknown to the attacker E. Generating it randomly is impossible due to the huge key space. If the attacker E attempts to substitute a random IdX, encrypt it with user's

D public key ($M3' = \text{EpubKd}(\text{IdX} + t1)$) and send it, the attack will immediately fail at the next stage.

Part 2. Failure of Checks at Step 5 (user D Level)

Threat: Assume that attacker E knows the identifier IdC and was able to transmit data $M3'$ containing modified or false data to user D, attempting to bypass the final verification logic.

Having received the forged packet $M3'$, user D decrypts it with his private key at Step 5 and extracts the components: $\text{IdC}^* + t1^* = \text{EprivKd}(M3')$. Next, user D fulfills two strict mandatory verification conditions:

Condition 1 (Session Freshness): The extracted timestamp $t1^*$ must exactly match the timestamp $t1$ from their own initial request at Step 2 ($t1^* = t1$). If E used an old intercepted packet, the verification will block the attack.

Condition 2 (Preimage Validity): The hash of the extracted identifier must be equal to the hash $H2$ that user D sent to the contract in Step 2: $\text{Hash}(\text{IdC}^*) = H2$.

Final Barrier: In Step 2, user D determines the hash $H2 = \text{Hash}(\text{IdC})$. At this same step, the smart contract verifies that $H2$ matches the on-chain hash $H1$ originally provided by the data author (user A). If the data belongs to someone else or is compromised, then $\text{Hash}(\text{IdC}^*) \neq H1$. Any attempt by attacker E to supply an invalid IdC^* will violate the equality $\text{Hash}(\text{IdC}^*) = H2$, and the protocol will be terminated.

Conclusion: $\square$

General conclusion regarding Attack 2: Protocol 2 is completely secure against verification falsification. Attacker E is physically unable to bypass the audit, as user's D final verification strictly binds three components: the off-chain timestamp $t1$, the verifiable identifier IdC, and the immutable on-chain hash $H1$. Any discrepancy in this chain makes the forgery obvious.

To prove the resilience of Protocol 2 to Attack 3 (Replay Attack), we will mathematically and logically justify why reuse of a previously intercepted valid data $M3'$ by an attacker E is guaranteed to fail. The protocol's security in this scenario is based entirely on the cryptographic binding of the dynamic time parameter $t1$ to the verified identifier within the isolated ciphertext.

Theorem 10. *Protocol 2 is resilient to the replay attack: an unauthorized user cannot reuse a previously intercepted message $M3$.*

Proof. **Threat.** Attacker E passively intercepted a valid data $M3_{\text{old}}'$ during a previous successful audit session. After some time, user D initiates a new audit session (Step 2). Attacker E intercepts traffic and, in Step 5, slips user D an old $M3_{\text{old}}'$ packet, attempting to conceal the fact that the blockchain data has been modified, deleted, or that user A is no longer a valid participant.

Initiating a new session (Step 2): User D generates a new, currently valid timestamp, $t1_{\text{new}}$, and sends it to the smart contract. The $t1_{\text{new}}$ parameter is a unique marker for the current session (nonce/timestamp).

Structure of the intercepted attacker packet: The $M3_{\text{old}}$ data, in the attacker's possession, was generated by user A in Step 4 in the past. The mathematical structure is strictly fixed: $M3_{\text{old}} = \text{EpubKd}(\text{IdC} + t1_{\text{old}})$, where $t1_{\text{old}}$ is the obsolete timestamp from the previous session, and $t1_{\text{old}} \neq t1_{\text{new}}$.

User's D actions in Step 5: After receiving $M3_{\text{old}}$ from the attacker, user D uses his private key privKd to decrypt the data: $\text{IdC}^* + t1^* = \text{DprivKd}(M3_{\text{old}}) = \text{IdC} + t1_{\text{old}}$.

Logical verification barrier: Immediately after decryption, user D performs the first mandatory check from the conjunction of conditions in Step 5: $t1^* = t1_{\text{new}}$. By substituting

actual values, the system verifies the equality $t1_{old} == t1_{new}$. Since the timestamp inside the packet refers to the past, the equality is violated: $t1_{old} \neq t1_{new}$.

Conclusion: The protocol immediately terminates on user's D side. For the replay attack to succeed, the attacker must change the old value inside the packet to the current one, i.e., convert $t1_{old}$ to $t1_{new}$. However, the packet is encrypted with user's D public key $pubKd$. Under the IND-CCA2 asymmetric encryption security assumption, attacker E cannot decrypt $M3_{old}$ without the private key $privKd$. Attempting to blindly change the bits in the ciphertext (without decrypting) due to the avalanche effect of the cryptographic algorithm will result in user D receiving destructured random noise instead of valid data upon decryption, and the protocol will still fail. $\square$

General conclusion regarding Attack 3: Protocol 2 is completely resistant to replay attacks. The use of dynamic timestamps $t1$ as off-chain nonces sealed within the asymmetric ciphertext makes it impossible to use old responses in new sessions. An attacker E cannot disguise the modification or deletion of data.

We summarize all the protective mechanisms of Protocol 2 in Table 7.

Table 7. Safeguards of Protocol 2.

Point of Attack	Protocol Protective Barrier	Result for the Offender
Blocking Matrix for Attack 1		
Passive Interception and Decryption: The attacker intercepts an encrypted packet M1 from the network and attempts to conduct a chosen ciphertext attack to extract the message DataC and find out who it belongs to IdC.	Strength of asymmetric encryption (IND-CCA2) on the key of user A $pubKa$.	The array remains unreadable. Without $privKa$ private key, the mathematical complexity of decryption is equivalent to a brute-force attack.
Brute-force attack (Dictionary attack): The attacker reads the public hash H1 from the blockchain and tries to enumerate possible IdX values of users off-chain, hash them, and compare them with H1 to deanonymize user C.	Preimage Resistance of a One-Way Hash Function.	With sufficient entropy of the identifier (a long random string), the computational complexity is 2^{256} operations, which is unrealistic.
Off-chain traffic interception: An external attacker E intercepts data M3 as it is transmitted from the smart contract to user D, attempting to eavesdrop on the verification result and extract the IdC hidden inside.	Strength of asymmetric encryption (IND-CCA2) on the key of the verifying user D $pubKd$.	Attacker E sees only pseudo-random noise bytes. Extracting the data without the private key $privKd$ is impossible.
Privilege Escalation Attempt: The legitimate verifier D successfully decrypts the received data M3 with its key $privKd$ and attempts to find in it the original secret data DataC, to which it initially does not have the access flag.	Architectural isolation of business logic (complete exclusion of the DataC variable from the Step 4 algorithm.	User D sees only the IdC confirmation and the $t1$ timestamp. The actual DataC data is not physically present inside the packet, as user A did not include it.

Table 7. Cont.

Point of Attack	Protocol Protective Barrier	Result for the Offender
Blocking Matrix for Attack 2		
False Data Injection: The attacker attempts to intercept the request in Step 3, impersonate user A, and send a blindly generated ciphertext M3' to the contract to confirm the validity of the non-existent data C.	Difficulty of choosing the correct IdC (Identity of a hash function to search for a preimage).	Attacker E doesn't know the real IdC locked in the blockchain. Sending a random IdX will invalidate the data.
Spoofing Attack: An attacker attempts to spoof a timestamp or substitute someone else's IdY to convince user D that another user is logged in to the system, posing as user C.	Strict conjunctive condition to check in Step 5: $(t1' = t1) \& (\text{Hash}(\text{IdC}') = H2)$.	Changing the timestamp or spoofing the ID destroys the mathematical equations on user's Dside. The protocol is immediately terminated.
Blocking Matrix for Attack 3		
Old Session Injection (Replay): The attacker blocks user's A fresh response and sends user D a previously recorded M3 _{old} packet to simulate a successful audit of deleted/modified data.	Cryptographically anchor the timestamp within the ciphertext under the pubKd key.	The retrieved timestamp $t1_{old}$ does not match the expected current timestamp $t1_{new}$. The protocol is terminated.
Bit-flipping attack: An attacker attempts to modify the structure of an encrypted M3 _{old} packet on the fly to adjust the old timestamp to a new one without decrypting the entire packet.	Resistance of an asymmetric cipher to modification of the ciphertext (IND-CCA2 integrity property).	During decryption, user D receives corrupted data (garbage). Hash and time checks are guaranteed to fail.

2.10. Security Proof for Protocol 3

Protocol 3 describes a three-pass encryption scheme adapted to the blockchain environment for securely exchanging public keys (pubKa, pubKb) without revealing them to a passive observer. Let's formulate the attacker's goals and attack vectors.

Attack 1: Deanonymization and key disclosure (Violation of exchange confidentiality). The attacker, having intercepted the public transformation chain from the blockchain (e.g., for party A: $Xa \rightarrow Yb \rightarrow Za$), attempts to mathematically calculate the original public key pubKa or the secret masking keys kA, kB.

Attack 2: Man-in-the-Middle attack at the address level. The attacker E attempts to intercept the value Xa in Step 2, replace it with their own value Xe, and, at the end of the protocol, trick user B into accepting the false public key pubKe instead of pubKa.

To prove the robustness of Protocol 3 to Attack 1 (Deanonymization and Key Discovery), we mathematically justify why a global passive attacker E, having intercepted the full on-chain transformation chain, cannot compute the original public keys pubKa and pubKb or the secret masking factors kA and kB. The protocol uses a hybrid approach. The blockchain infrastructure is derived from Ethereum's elliptic curve, but the masking algorithm itself operates in a multiplicative residue group modulo $p \in \mathbb{Z}p^*$, where the base element is the X coordinate of the public key (in compressed format), and the operations are classical exponentiation modulo p.

Theorem 11. *Protocol 3 is resistant to deanonymization and key disclosure: an unauthorized user who intercepts the complete on-chain chain of transformations will not be able to calculate the original public keys $pubKa$ and $pubKb$ or the secret masking factors kA and kB .*

Proof. Mathematical context: p is a fixed large prime number (the order of the underlying field of the curve used in the blockchain). The transmitted secret is the number $pubKa$, which represents the shorthand for the elliptic curve point of user's A public key, where $pubKa \in \mathbb{Z}_p^*$. Users independently generate large random numbers kA and kB that are coprime to $(p - 1)$. Using the extended Euclidean algorithm, the multiplicative inverses of kA^{-1} and kB^{-1} are calculated, such that $k \times k^{-1} \equiv 1 \pmod{(p - 1)}$ (according to Fermat's Little Theorem).

Data Tracing in the Blockchain. An attacker E passively intercepts the transaction chain for party A from the blockchain. The following set of numbers in the $\mathbb{Z}_p^*$ field becomes available to him:

$$\text{Step 1: } Xa = (pubKa)^{kA} \pmod{p};$$

$$\text{Step 3: } Ya = Xa^{kB} = (pubKa)^{kA \times kB} \pmod{p};$$

$$\text{Step 5: } Za = (Ya)^{kA^{-1}} = (pubKa)^{kA \times kA^{-1} \times kB} = (pubKa)^{kB} \pmod{p}.$$

The set of public data for party A is a tuple: $TA = \{Xa, Ya, Za\}$.

Constructing a Reduction Game. Suppose there is an attacker E who possesses the Crack(TA) algorithm, which can compute the original value ($pubKa$) or secret exponents (kA , kB) from an intercepted tuple in polynomial time. Let's construct a simulator Sim, that uses this Crack() "black box" to solve the classical discrete logarithm problem (DLP) modulo p :

- ✓ The DLP Challenger feeds the simulator Sim two numbers: the known base Y and the result of raising to the power V , where $V = Y^x \pmod{p}$. The Sim's task is to find the secret exponent x .
- ✓ The simulator Sim initiates the attacker E and constructs a fake but mathematically indistinguishable Protocol 3 session:
 - The simulator Sim declares the value Y to be the target key $pubKa$.
 - The simulator Sim declares the result V (which is equal to Y^x) to be the first-step element Xa . Thus, the hidden exponent x implicitly becomes the masking key kA (i.e., $kA = x$).
 - Next, the simulator Sim generates a random number kB^* (coprime to $(p - 1)$) and calculates the remaining elements of the chain according to the rules of Protocol 3:
 - $Ya^* = (Xa)^{kB^*} \pmod{p} = (B^x)^{kB^*} \pmod{p};$
 - $Za^* = (pubKa)^{kB^*} \pmod{p} = B^{kB^*} \pmod{p}.$

Data transmission: The simulator Sim transmits the generated tuple $T^* = \{V, Ya^*, Za^*\}$ to the attacker E. To the attacker, this data looks completely standard: three numbers, where the first is raised to the power of kA , the second is raised to the power of $kA \times kB$, and the third is raised to the power of kB .

Response: Attacker E runs their algorithm and, by assumption, successfully cracks the protocol. To produce a result, they must calculate the masking keys. They return to the simulator Sim the value of the first-step key they found—the exponent to which the base was raised.

Final: The attacker returns the value of kA to the simulator Sim, which is exactly equal to x . Simulator B takes this number, passes it to Challenger, and wins the DLP game.

Contradiction. Since the modulus p in Ethereum is a huge 256-bit number, the DLP in the multiplicative group is computationally intractable in polynomial time (requiring about 2^{128} sieve operations). Therefore, the simulator Sim cannot exist, and therefore, an

attacker E has no mathematical ability to calculate the masking keys or the original key pubKa from the public blockchain logs. $\square$

General conclusion regarding Attack 1: The modified interpretation confirms that Protocol 3 is mathematically secure against passive deanonymization. The security of public key exchange is fully guaranteed by the fundamental mathematical complexity of the DLP in a multiplicative residue group modulo p .

To prove the resistance of Protocol 3 to the Man-in-the-Middle attack, we must analyze what happens in the mathematical apparatus of a three-pass exchange if an adversary E intercepts the value X_a in Step 2 and replaces it with their own value $X_x = (\text{pubKx})^{k_E} \bmod p$. The proof is based on strict algebraic rules for multiplicative groups and shows that this attack will inevitably lead to the destruction of the mathematical consensus (destructuring of the final value) in Step 6, as a result of which user B is guaranteed to detect the tampering and reject the spoofed key.

Theorem 12. *Protocol 3 is resistant to a Man-in-the-Middle attack: an unauthorized user cannot spoof values during data exchange.*

Proof. Let's consider step-by-step how the variables within the multiplicative group $\mathbb{Z}_p^*$ are modified when E attempts to inject a false value.

Initial state and dilemma of Step 2. User A sent $X_a = (\text{pubKa})^{k_A} \bmod p$ to the blockchain. Attacker E intercepts Step 2 and, instead of X_a , sends user B his own false value: $X_x = (\text{pubKx})^{k_E} \bmod p$, where pubKx is the false public key and k_E is the attacker's secret mask.

Manipulations in Step 3 (Actions of user B). User B is unaware of the spoofing. According to the protocol rules, they take the incoming value (thinking it is X_a) and convert it to their own secret masking key k_B . The output value Y_a^* is generated and sent to the blockchain: $Y_a^* = (X_x)^{k_B} = ((\text{pubKx})^{k_E})^{k_B} = (\text{pubKx})^{k_E \times k_B} \bmod p$.

Manipulations in Steps 4 and 5 (Actions of user A). The blockchain returns the Y_a^* value to user A. According to the protocol rules, user A is required to remove his mask by raising the received value to the power of k_A^{-1} : $Z_a^* = (Y_a^*)^{k_A^{-1}} = ((\text{pubKx})^{k_E \times k_B})^{k_A^{-1}} = (\text{pubKx})^{k_E \times k_B \times k_A^{-1}} \bmod p$. The resulting Z_a^* value is sent back to the blockchain.

Final step: Step 6 (Attempt to restore by user B). User B receives Z_a^* and attempts to remove his mask k_B^{-1} to obtain user's A key: $\text{pubKa}' = (Z_a^*)^{k_B^{-1}} = ((\text{pubKx})^{k_E \times k_B \times k_A^{-1}})^{k_B^{-1}} \bmod p$. Due to the commutativity of multiplication in the exponent and the multiplicative inversion rule $k_B \times k_B^{-1} \equiv 1 \bmod (p-1)$, the exponents k_B and k_B^{-1} cancel each other. The resulting equation is: $\text{pubKa}' = (\text{pubKx})^{k_E \times k_A^{-1}} \bmod p$.

Algebraic proof of attack failure. For the Man-in-the-Middle attack to be considered successful, user B must obtain a valid key from the attacker in Step 6, i.e., the identity $\text{pubKa}' = \text{pubKa}$ must hold. Let's substitute the resulting value of pubKa' into this equation: $(\text{pubKx})^{k_E \times k_A^{-1}} \equiv \text{pubKa} \bmod p$. This equality is mathematically possible only in one of two cases.

Case 1. A trivial key is found, $\text{pubKx} = 1$ or $\text{pubKx} = 0$, which is immediately rejected by the blockchain validator, since the X coordinate of a real point on an elliptic curve cannot take such values.

Case 2. The exponent is equal to one: $k_E \times k_A^{-1} \equiv 1 \bmod (p-1)$. Multiply both sides of the equation by k_A : $k_E \equiv k_A \bmod (p-1)$.

Conclusion of the reduction: For an attacker E to successfully impose his public key pubKx , his secret masking key k_E must be strictly equal to the secret masking key k_A of user A. However, according to the assumption, user A generates his key k_A completely randomly, independently, and stores it off-chain. This key is unknown to attacker E. The

probability of guessing a 256-bit random number k_A is equal to 2^{256} , which is computationally infeasible. If $k_E \neq k_A$, then user B in Step 6 receives not the key pubK_x , but the destructured “garbage” number $(\text{pubXk})^{k_E \times k_A - 1} \bmod p$. $\square$

General conclusion on Attack 1: Because the Protocol 3 requires strict, alternating application and removal of masks by both parties $k_A \rightarrow k_B \rightarrow k_A^{-1} \rightarrow k_B^{-1}$, an attacker cannot simply “break in” with his mask k_E . Their intervention breaks the inversion chain, and instead of the substituted key, the recipient is guaranteed to calculate a pseudo-random set of bytes.

Table 8 lists all the considered security mechanisms of Protocol 3.

Table 8. Safeguards of Protocol 3.

Point of Attack	Protocol Protective Barrier	Result for the Offender
Blocking Matrix for Attack 1		
An attempt to find the exponent k_A , knowing the base pubK_a and the result X_a .	Discrete logarithm problem (DLP)	It is impossible to calculate the degree, the data is protected by the strength of the one-way function.
Attempt to compute the shared session secret or split k_A and k_B .	Computational Diffie–Hellman problem (CDHP) over numbers	Overlaying two independent secret degrees completely hides the connection between transactions.
Trying to match Z_a and X_a to find the key relationship.	Uncertainty of a system of equations in a residue field.	The attacker sees one equation with two unknowns pubK_a and k_B , there are infinitely many solutions
Blocking Matrix for Attack 2		
Substitution of the legitimate first mask X_a with its own counterfeit X_x	Violation of end-to-end commutativity of three-pass mask removal	User’s A masks k_A^{-1} and the attacker’s mask k_E cannot mutually destroy each other at Step 5
Attempting to force user B to accept pubK_x as user’s A trusted public key.	Unsolvability of the equality $k_E \equiv k_A \bmod (p - 1)$ without knowledge of the secret k_A .	User B receives mathematical garbage (a random number) as output, which is guaranteed not to pass authentication.

2.11. Security Proof for Protocol 4

In Protocol 4, the primary goal of attacker E is to violate confidentiality (obtain the plaintext of message C or the IdC) or to violate integrity (disinformation of user B by imposing false data). Let’s consider possible attack scenarios.

Attack 1: Direct Unauthorized Access (Smart Contract Bypass)

Scenario: Attacker E (who does not have access to the IdC) wants to read message C.

Method: Attacker E does not invoke the smart contract but directly reads the data M1 and the hashes H1, H* from the public blockchain ledger where they were written in Step 1. Attacker E then attempts to crack the ciphertext M1 off-chain.

Attack 2: Identity Spoofing/ Access Flag Bypass (On-chain)

Scenario: Attacker E attempts to force a smart contract to provide them with a dataset in Step 3.

Method: In Step 2, attacker E sends a transaction to the contract, specifying a third-party IdB in the request body. He attempts to trick the contract into believing the request originates from legitimate user B.

Attack 3: Data Modification in Transit (Off-chain)

Scenario: Attacker E wants to mislead user B into accepting false data.

Method: Attacker E waits for the legitimate user B to make a valid request in Step 2. In Step 3, the contract sends a set $\{M1, H1, H^*\}$ to user B. Attacker E intercepts this off-chain set online and replaces the encrypted message M1 with a fake message $M1^*$.

Attack 4: Replay Attack

Scenario: Attacker E wants to force user B to accept an older version of message C (possibly deleted or modified by author A).

Method: User B initiates Step 2 for a new verification. Attacker E intercepts the contract's response in Step 3 and replaces the current set $\{M1, H1, H^*\}^{new}$ with an older set $\{M1, H1, H^*\}^{old}$, saved from previous transactions.

To prove the resistance of Protocol 4 to Attack 1 (Direct Unauthorized Access bypassing the Smart Contract), we will mathematically justify why an attacker E, with full passive access to the public blockchain ledger, cannot extract the content of message C from the public data. The proof is constructed within the framework of the game-theoretic model of the provable security method and boils down to the fundamental resistance of the symmetric encryption scheme used to chosen-plaintext attacks (IND-CPA).

Theorem 13. *For Protocol 4, an unauthorized user cannot extract the content of message C from the public data.*

Proof. Mathematical Context and Conditions. K_{ab} is a secret symmetric key known exclusively to legitimate users A and B (generated as a result of applying Protocol 3). In Step 1, the set $T = \{M1, H1, H^*\}$ is written to the public blockchain, where $M1 = Enc_{K_{ab}}(C)$ is the result of symmetric encryption of message C. The hashes $H1 = Hash(IdC)$ and $H^* = Hash(C)$ are obtained using a cryptographically strong one-way function.

We perform a reduction to the IND-CPA strength of a symmetric cipher.

Analysis of the attacker's data distribution. The attacker E completely ignores the smart contract and reads the set T directly from the blockchain node. He then has access to two hashes and one ciphertext. Suppose an attacker E has a polynomial-time algorithm $Crack(M1, H1, H^*)$ that allows him to crack the ciphertext M1 with a significant (non-negligible) advantage and recover the plaintext message C without knowing the key K_{ab} .

Constructing a Reduction Game. Let's construct a simulator Sim whose task is to use an attacker's algorithm E to win the standard cryptographic game IND-CPA (Indistinguishability of Ciphertexts under Chosen Plaintext Attacks) against the Challenger of a symmetric cipher:

- ✓ The Challenger of the IND-CPA game implicitly generates a random secret key K^* and keeps it secret.
- ✓ The simulator Sim selects two test messages of equal length, $C0$ and $C1$, and sends them to the Challenger.
- ✓ The Challenger selects a random secret bit $b \in \{0, 1\}$, encrypts the selected message with the secret key, and returns to the simulator the target ciphertext $M^* = Enc_{K^*}(C_b)$.
- ✓ Now the simulator Sim must launch the attacker E and simulate Step 1 of Protocol 4 for it:
 - The simulator declares M^* to be the intercepted array M1.
 - The simulator independently calculates the hashes for one of the messages (for example, for $C0$): $H1^* = Hash(IdC0)$ and $(H^{**} = Hash(C0))$.
 - The simulator transmits the tuple $T^* = \{M^*, H1^*, H^{**}\}$ to the attacker E.
- ✓ The attacker E runs its $Crack()$ algorithm, cracks the ciphertext M^* , and returns the decrypted message text C_{result} to the simulator Sim.

- ✓ The simulator Sim verifies the response: if ($C_{result} = C_0$), the simulator sends the Challenger the assertion that bit $b = 0$; otherwise, $b = 1$. The simulator wins the IND-CPA game with a probability equal to the attacker's success.

Obtaining a Contradiction. According to the assumption, the symmetric encryption algorithm used (e.g., AES-256 in GCM mode) is strictly IND-CPA-secure (and even IND-CCA2-secure). This means that it is mathematically impossible to construct a simulator capable of guessing bit b with an advantage significantly exceeding random distribution ($1/2 + \epsilon$). Therefore, our assumption is incorrect: the attacker E does not have a Crack() algorithm.

The Role of the Hash Functions H1 and H* in the Context of Attack 1. Attacker E also sees the values $H1 = \text{Hash}(\text{IdC})$ and $H^* = \text{Hash}(C)$ in the blockchain. Due to the one-wayness property of the Keccak-256/SHA-256 hash function, the attacker cannot mathematically reverse H1 to discover the network address of IdC. As defined earlier, the address has maximum entropy (a pseudorandom 160/256-bit string), which completely eliminates the possibility of a successful brute-force attack. The hash H^* is an exact snapshot of the message C. If the message C were short (for example, the word "Yes" or "No"), an attacker could try the possible entries, hash them, and compare them with H^* . To prevent this, Protocol 4 assumes high entropy of the message C by default (the presence of unique data, salt, or a random initialization vector (IV) within the cipher). Without knowledge of the plaintext, running a dictionary attack on the hash H^* is computationally infeasible. $\square$

General conclusion regarding Attack 1: Protocol 4 is completely resistant to direct unauthorized access via blockchain reading. The public status of the ledger does not compromise confidentiality, as the message content C is completely isolated by the symmetric key K_{ab} , and the identifiers are hidden by one-way hashing. Attacker E is left with only a set of cryptographically private data.

To prove resistance of Protocol 4 to Attack 2 (Identity Spoofing/ Access Flag Bypass), we must mathematically and logically justify why attacker E cannot deceive the smart contract in Step 2, even if they explicitly specify someone else's legitimate identifier IdB in their transaction body. This proof is based on the fundamental architecture of blockchain authentication and the isolation logic of the virtual machine EVM execution environment.

Theorem 14. *For Protocol 4, an unauthorized user cannot use someone else's legitimate identifier IdB within the body of their transaction.*

Proof. Mechanism for submitting a transaction to the blockchain. In accordance with Step 2 of the protocol, an attacker E creates a call to the smart contract. To do this, he generate a transaction, signing it with his personal private key privKe . The attacker's transaction contains:

- ✓ Data: the IdB argument (the forged identity) and the hash $H2 = \text{Hash}(\text{IdC})$.
- ✓ Cryptographic signature of the transaction: $\text{SprivKe}(\text{Data})$.

Verification at the blockchain consensus level (Entry barrier). For each user of the blockchain system, an address on the blockchain network is hard-coded from the public key. For example, for Ethereum: $\text{AddressE} = 0x + \text{LSB}_{20}(\text{Keccak-256}(\text{pubKe}))$. This address is assigned to the `msg.sender` system variable within the smart contract environment. An attacker cannot change or tamper with the value of `msg.sender`, as doing so would disrupt the consensus of the entire network.

Validation logic within the smart contract (Step 2). Upon receiving a request, the smart contract performs three sequential checks:

Address and key match: The contract verifies that the received network address matches the account's public address (the last 20 bytes of the Keccak-256 hash of the public key with the 0x prefix). For the attacker, this equality will be true, since their address, AddressE, is honestly derived from his key, pubKe.

Data presence check: The contract verifies that ($H2 = H1$). Assuming the attacker knows this hash, the check is passed.

Access flag check (Critical barrier): The contract reads the access flags that user A set in Step 1. According to the terms, access is allowed only to the legitimate user B (AddressB). Since the transaction was signed by the attacker E, the msg.sender variable contains AddressE. The identity $\text{AddressE} = \text{AddressB}$ is false: $\text{AddressE} \neq \text{AddressB}$.

Mathematical impossibility of bypassing by address forgery. For a spoofing attack to succeed, attacker E must force the system to output $\text{msg.sender} = \text{AddressB}$. They have only two options, both of which are cryptographically infeasible:

Path A (Forging a transaction signature): The attacker must sign the transaction with someone else's key privKb . According to the EUF-CMA property of the ECDSA, it is impossible to calculate the private key from a known address or public key (the difficulty of the elliptic curve discrete logarithm problem (ECDLP)).

Path B (Finding an address collision): The attacker must generate a new key pair that will yield exactly the same address as user B: $\text{LSB}_{20}(\text{Keccak-256}(\text{pubKe_new})) = \text{AddressB}$. The address length is 160 bits. The computational complexity of finding the first preimage for Keccak-256 is 2^{160} operations, which is absolutely infeasible in polynomial time. $\square$

General conclusion regarding Attack 2: Protocol 4 has absolute on-chain resistance to identity spoofing. The smart contract makes the decision to issue a dataset at Step 3 not based on the text or digital IdX written in the transaction, but on the sender's unforgeable cryptographic address (msg.sender), which is tightly linked to the transaction's electronic signature. Attacker E is guaranteed to be blocked at Step 2, and the protocol terminates.

To prove the robustness of Protocol 4 to Attack 3 (Data Modification in the Transit Channel/Integrity Violation), we will mathematically justify why an attacker E cannot surreptitiously inject false data. Even if the adversary completely controls the communication channel at Step 3 (Dolev-Yao model) and replaces the original off-chain packet, the attack is guaranteed to fail at the final stage due to the strict conjunctive hash verification condition on the receiving side.

Theorem 15. *For Protocol 4, an unauthorized user cannot surreptitiously inject false data.*

Proof. Initial state of parameters (Step 1). In Step 1, user A commits the following reference values to the immutable blockchain ledger:

- ✓ $M1 = \text{Enc}_{K_{ab}}(C) = \text{Enc}_{K_{ab}}(\text{DataC} + \text{IdC});$
- ✓ $H1 = \text{Hash}(\text{IdC});$
- ✓ $H^* = \text{Hash}(C) = \text{Hash}(\text{DataC} + \text{IdC}).$

Actions of Attacker E in Step 3. When the smart contract sends a packet $\{M1, H1, H^*\}$ to user B, the attacker intercepts it and replaces the ciphertext $M1$ with a falsified $M1^*$. The packet arriving to user B takes the form: $\{M1^*, H1, H^*\}$. The attacker is forced to leave the hashes $H1$ and H^* unchanged, as they must strictly correspond to the standards already recorded in the blockchain in Step 1; otherwise, user B will immediately detect the substitution when checking the ledger.

Decryption on user's B Side. After receiving the spoofed packet, user B applies the shared secret key K_{ab} to the spoofed data $M1^*$: $\text{DataC}^* + \text{IdC}^* = \text{Dec}_{K_{ab}}(M1^*)$.

Due to the avalanche effect and the properties of symmetric encryption algorithms (e.g., AES or Kuznyechik), decrypting the spoofed ciphertext $M1^*$ with the correct key Kab will lead to one of two results:

Case A (Authenticated modes, e.g., AES-GCM): The decryption algorithm will immediately return an integrity error (authentication tag validation error), and the protocol will terminate.

Case B (Classical modes, e.g., AES-CBC): The output will be a completely pseudorandom byte string (mathematical “garbage”), where $DataC^* \neq DataC$ and $IdC^* \neq IdC$.

Final barrier: Hash verification (Step 3). User B satisfies the mandatory end-to-end verification condition: $(H1 = Hash(IdC^*)) \ \& \ (H^* = Hash(DataC^* + IdC^*))$. Substitute the resulting “garbage” byte string into this equation. For attacker E to be deceived, two identities must simultaneously hold:

- ✓ $Hash(IdC^*) == H1$ (where $H1 = Hash(IdC)$);
- ✓ $Hash(DataC^* + IdC^*) == H^*$ (where $H^* = Hash(DataC + IdC)$).

Reduction to the Second Preimage Resistance of a Hash Function. To satisfy these equalities, given that the extracted data is not equal to the original ($IdC^* \neq IdC$) and ($DataC^* \neq DataC$), attacker E must solve the Second Preimage Resistance problem for the cryptographic hash function. Under this assumption, the hash function used is absolutely resistant to the Second Preimage Resistance problem. For 256-bit hashes (SHA-256/Keccak-256), the computational complexity of finding such a match is 2^{256} operations. This problem is mathematically unsolvable in polynomial time. If even one bit in $M1^*$ changes, the hashes computed by user B will completely change due to the avalanche effect, the equality will be violated, and the protocol will immediately terminate with an error status. $\square$

General conclusion regarding Attack 3. Protocol 4 is resistant to data modification in communication channels. Despite the fact that the dataset is transmitted from the smart contract to user B in Step 3 in an unprotected off-chain environment, attacker E is unable to alter the information. Protection is ensured by the fact that the reference data hashes $H1$ and H^* are pre-established in the blockchain, and the avalanche effect of the hash function makes any off-chain modification of the ciphertext obvious upon the first verification on the recipient’s side. The protocol fully guarantees integrity.

Theorem 16. *Protocol 4 is resistant to replay attacks.*

Proof. Initial state of the system (Update of record by author A). Suppose user A decides to update or invalidate message C. In Step 1, he sends a new, valid packet to the blockchain: $T_{new} = \{M1_{new}, H1_{new}, H_{new}^*\}$. Thanks to blockchain consensus, the old record is overwritten in the smart contract’s state variable, or the contract begins referencing the new, valid hash.

Request from user B (Step 2). User B initiates a new verification session. The smart contract reads the current T_{new} data from its current, secure on-chain storage and sends it in Step 3.

Actions of Attacker E. Attacker E intercepts the off-chain transit of Step 3 and completely replaces the incoming packet with the old stored set: $T_{old} = \{M1_{old}, H1_{old}, H_{old}^*\}$.

Attack failure on user’s B side (end-to-end on-chain verification). In the architecture of hybrid blockchain protocols, user B, upon receiving off-chain data, is required to verify it against the current on-chain reference. The process occurs as follows:

- ✓ User B independently reads the current hash values $H1$ and H^* stored in the ledger at that time.
- ✓ The contract returns the on-chain values $H1_{new}$ and H_{new}^* to him.

- ✓ User B compares the hashes from the T_{old} packet received from the attacker with those returned by the blockchain: $(H1_{old} = H1_{new}) \& (H_{old}^* = H_{new}^*)$?
- ✓ Since User A modified the data, the hashes are guaranteed to disagree ($H1_{old} \neq H1_{new}$) and ($H_{old}^* \neq H_{new}^*$).

Conclusion: If the author updated the information in the blockchain, the blind replay of the old packet in the communication channel is immediately detected by the recipient. The protocol terminates with an error status. $\square$

General conclusion regarding Attack 4. Protocol 4 is resistant to replay attacks if the data was previously updated by the author on the blockchain. The persistence of the distributed ledger serves as a source of truth: the recipient can always verify that the data sent from the off-chain channel corresponds to the current on-chain consensus.

Table 9 lists all the considered defense mechanisms of Protocol 4.

Table 9. Safeguards of Protocol 4.

Point of Attack	Protocol Protective Barrier	Result for the Offender
Blocking Matrix for Attack 1		
Attempting to decrypt off-chain or perform cryptanalysis on an intercepted packet.	IND-CPA strength of a symmetric cryptographic algorithm	The data is indistinguishable from random noise. Without the Kab key, C cannot be read.
Attempt to recover IdC network address from its hash image.	The hash function is resistant to preimage searches + the blockchain address has high entropy	The problem is mathematically unsolvable in polynomial time.
Attempt to run a dictionary search (Brute-force) to reveal the content of the text.	One-way hash function.	The avalanche effect prevents partial selection of information.
Blocking Matrix for Attack 2		
Injecting someone else's IdB identifier into the transaction body in order to spoof access flags.	Automatically committing msg.sender at the blockchain virtual machine level	The contract sees the real address of E, not user B. Flag checking is not performed.
Generating a fake private key to impersonate user's B address.	The resistance of Keccak-256 to the search for a preimage is 2^{160} computational operations).	It is mathematically impossible to guess a key for someone else's address. The transaction is rejected by the registry.
Blocking Matrix for Attack 3		
Modification of bits or complete substitution of the encrypted array $M1 \rightarrow M1^*$ "on the fly" during packet transit.	Avalanche effect of symmetric decryption on the Kab key	The recipient extracts destructured "garbage" from the packet instead of legitimate data.
An attempt to force user B to accept corrupted data as genuine.	The resistance of a hash function to searching for the second preimage is 2^{256} operations.	The calculated hashes from the "garbage" data do not match the on-chain benchmarks. The protocol terminates.
Blocking Matrix for Attack 4		
Replacing the current package with the old T_{old} in the Step 3 channel.	Comparison of hashes of the incoming packet with the current state variables of the smart contract.	The H_{old} hash does not match the on-chain H_{new} benchmark.

3. Results and Discussions

3.1. Performance Evaluation of Various Hybrid Cryptographic Algorithms

For Protocol 4, a combination of different algorithms must be used: generating a shared secret key using an asymmetric elliptic curve cipher and encrypting personal data using a symmetric encryption algorithm. The study assessed the performance of various hybrid cryptographic algorithms for encrypting 32 bytes, 1 MB, and 10 MB of data for subsequent storage in the Ethereum and Hyperledger Fabric blockchain platforms. The testing included the EC + El-Gamal scheme, as well as ECDH bundles with symmetric AES ciphers (in the software and hardware-accelerated AES-NI implementation), Magma (CTR/CBC), and Grasshopper (CTR/CBC). Direct asymmetric encryption of EC + El Gamal demonstrated unacceptable time for practical use on large data (349.44 s per 10 MB), whereas hybrid ECDH-based schemes showed a linear increase in time with increasing volume.

A comparison of data encryption speeds utilizing different encryption methods is presented in Table 10.

Table 10. Data encryption speed for elliptic cryptography.

Algorithm	32 Bytes	1 MB	10 MB
EC + El Gamal	0.00117654 s	34.398 s	349.44 s
ECDH-AES	0.000001995 s	0.0270092 s	0.260765 s
ECDH-AES (AES-NI)	0.000000806 s	0.00196759 s	0.0202955 s
ECDH-Magma (CTR)	0.001 s	0.023 s	0.224 s
ECDH-Magma (CBC)	0.001 s	0.024 s	0.227 s
ECDH-Kuznyechik (CTR)	0.001 s	0.019 s	0.152 s
ECDH-Kuznyechik (CBC)	0.002 s	0.02 s	0.156 s

The obtained speeds' results allow us to recommend an architecture with a single ECDH key negotiation and subsequent symmetric encryption for Ethereum and Hyperledger Fabric. The highest speed was recorded for AES with hardware acceleration (0.0203 s per 10 MB), which is optimal for high-load systems. If it is necessary to comply with Russian standards, the Kuznyechik algorithm in CTR mode is preferable, providing 0.152 s per 10 MB, which is quite acceptable for off-chain encryption in Ethereum or execution inside chaincode in Hyperledger Fabric.

In the period of 2024–2025, the US National Institute of Standards and Technology (NIST) completed work on finalizing key standards for post-quantum cryptography (PQC). The first of the approved standards was ML-KEM (FIPS 203), a key encapsulation mechanism based on lattice mathematics, which was known previously as Kyber. The next important standard is ML-DSA (FIPS 204), which is a digital signature algorithm, also based on lattices and previously named Dilithium. The SLH-DSA standard (FIPS 205) completes this list, which describes a digital signature mechanism based on hash functions and known previously as SPHINCS+ [12–28].

Table 11 shows the results of the experiment of the encryption speed of post-quantum algorithms on various amounts of data. The characteristics of the computer on which the practical experiment was conducted are: DDR3 16GB RAM, Intel(R) Core(TM) i5-4460 CPU @ 3.20GHz, Ubuntu 24.04 (WSL). Table 11 presents the average data obtained as a result of a series of 10 experiments.

Table 11. Data encryption speed for post-quantum cryptography.

Algorithm	32 Bytes	1 MB	10 MB
ML-KEM-768 + ML-DSA + AES-256-GCM	0.129259 s	0.270072 s	1.241994 s
ML-KEM-768 + SLH-DSA + AES-256-GCM	1.59255 s	1.755355 s	3.612205 s

Table 12 presents a comparative characteristic for the classical and post-quantum schemes. The characteristics of the computer on which the practical experiment was conducted are: DDR3 16 GB RAM, Intel(R) Core(TM) i5-4460 CPU @ 3.20 GHz, Ubuntu 24.04 (WSL). Table 12 presents the average data obtained as a result of a series of 10 experiments.

Table 12. Comparative characteristics of the classical and post-quantum schemes for 32-byte data.

Algorithm (Size Sign)	Data + ID	M1/M2	S1/S2	Transact
ECDH-AES (AES-NI)	58 bytes	524 bytes	142 bytes	1226 bytes
ML-KEM-768 + ML-DSA + AES-256-GCM	58 bytes	3240 bytes	6618 bytes	10,415 bytes

The weight of a transaction in the registry (Ledger) increases by approximately 8.5 times (from 1226 B to 10,415 Bytes).

3.2. Implementation on the Ethereum Platform

The practical testing of the proposed protocols on the Ethereum platform has been implemented as a software package consisting of a decentralized smart contract and a client application with a graphical interface. The main purpose of this implementation is to ensure a strict separation between the public storage of metadata in a distributed registry and the local execution of resource-intensive cryptographic operations. The choice of the Ethereum platform is based on its well-developed ecosystem of smart contracts and support for standard libraries for interacting with the blockchain in Python (versions 3.11 and higher), such as web3.py. The Ganache environment has been used as a local testnet, which provides a deterministic environment for debugging transactions and monitoring registry status.

The architecture of the SecureTextExchange.sol smart contract is designed to minimize the computing load on the Ethereum Virtual Machine (EVM), thereby reducing gas consumption. The contract acts as a coordinator of interactions and a repository of encrypted data presented in the format of Base64 strings. The main data structure in the contract is a register of records, which records encrypted packets, hashes of subject identifiers, and digital signatures of the owners. An important component of the system is the mechanism for mandatory registration of users' public keys. Before starting work, each participant calls the registration function, which verifies the correspondence between the sender's Ethereum address and the uncompressed public key provided by them. This verification is delegated to the CryptoLib.sol auxiliary library, which calculates the address by taking the last twenty bytes of the Keccak-256 hash from the public key, which guarantees the authenticity of the identification of participants without disclosing their private keys.

The cryptographic module of the client application is implemented in a separate software layer crypto_utils.py, which integrates low-level primitives for working with the elliptic curve secp256k1. To ensure the confidentiality of personal data, an integrated elliptic curve encryption scheme (ECIES) is used. Within the framework of this scheme, an ephemeral key pair is generated for each encryption operation, on the basis of which a shared secret is calculated using the Diffie–Hellman algorithm (ECDH). The received secret is used as input data for the HKDF (HMAC-based Key Derivation Function) key derivation function, which forms a symmetric key for the AES-256 algorithm in GCM mode. This

multi-level approach provides authenticated encryption and protects data from attacks based on ciphertext modification. All generated data packets are signed by the owner using the ECDSA, while the signatures have a standard format of sixty-five bytes, including the *r*, *s*, and *v* components.

As part of the data encryption contract, the information recording process (the record-Data function) includes the transfer of encrypted M1 data, to which 64 bytes are added, allocated for two bytes32 hashes (H1 and H*), as well as the S1 signature of 65 bytes and the title. The access request operation (requestAccess function) is characterized by a fixed size of 116 bytes, which consists of an address, a public key (occupying 64 bytes), an H2 hash (32 bytes), a status, and a timestamp. The request approval procedure (the approveRequest function) consists of reencrypted M2 data and an S2 signature, which is 65 bytes long.

The system also provides for fixed overhead costs, which consist of several components. Firstly, it is a public key that occupies 65 bytes (prefix 0x04 plus 64 bytes of data). Secondly, a nonce of 12 bytes is used. Thirdly, the authentication tag of the AES-GSM algorithm takes up 16 bytes. In total, these constant overhead costs amount to 93 bytes. If we add the S1 signature (65 bytes) to this amount, then the total overhead for the write operation reaches 158 bytes.

Based on this data, the total volume is calculated: the percentage increase is calculated as the ratio of 158 bytes to the original data size multiplied by 100%, and the encrypted size itself is obtained by adding 158 bytes to the original volume. It is important to note that when calculating the size, the identifier, title, and personal data are taken into account, while the addresses of the sender and recipient are not included in this calculation.

As a result, the amount of text we can place is 30 kb (30,720 characters). The results of writing data to the blockchain are presented in Figures 5–7.

SecureTextExchange - Secure Exchange Protocol

Connection Data Recording Access Request Incoming Requests Receiving Data My Data Statistics

Step 1: Recording Encrypted Data to Blockchain

Data to Record

Subject Identifier (IDC): e.g.: user123 or passport

Title:

Personal Data:

Allowed Addresses (one per line):

Result (hashes and signatures)

☒ Data successfully recorded to blockchain!

Tx Hash: 369177fef6c193906e77ee8e277d96da71c10031290ae2ccba3cfcbee32d57ba

R1 (identifier): 0xalc2b8080ed4b6f56211e0295659ef97dd454b0a884198c10384f230525d4ee8

☒ Connected | 0xAE317018... | 99.9941 ETH

Figure 5. Record transactions in a public blockchain (Ethereum).

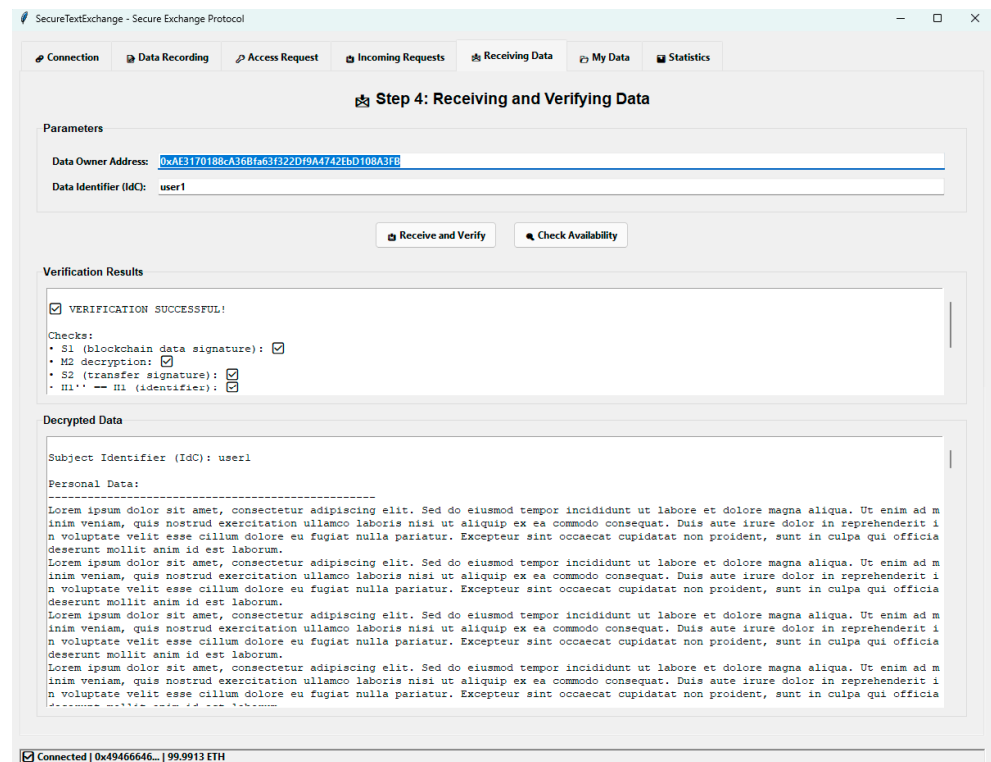


Figure 6. Receiving and Verifying Data (Ethereum).

The graphical interface of the application, built on the basis of the Tkinter library, ensures the consistent passage of all protocol steps through a system of functional tabs. The user is provided with tools to configure the connection to the node, manage key pairs, and monitor incoming data access requests. The access rights management tab allows the owner to dynamically change the list of authorized recipients for each specific record, which is implemented through modification of the corresponding mappings in the smart contract. The process of verifying the received data on the recipient's side includes a 5-fold verification, covering the validity of the sender's signatures, the integrity of the hash amounts, and the success of message decryption. With a positive result of all checks, the data is recognized as valid and displayed to the user, which eliminates the possibility of using fake or modified records from the blockchain.

#	Sender	Type	Data ID	Size (bytes)	Time	Tx Hash
1	0xAE3170...08A3FB	Record	0xa1c2b8080ed4b6...	41,349	2026-03-25 23:19:24	0x7cc309111bc7f12...
2	0x494666...CA7a1e	Request	0xa1c2b8080ed4b6...	116	2026-03-25 23:23:15	0x2e250740142092...
3	0xAE3170...08A3FB	Approval	0xa1c2b8080ed4b6...	41,277	2026-03-25 23:24:05	0x1a7a5ffbc8fc29...

Figure 7. Registry Transaction Statistics (Ethereum).

The security of this implementation is enhanced: the private key never leaves the client application, and it is not transferred to the blockchain network. All transactions are signed locally and sent to the network in an already generated form. Using the Keccak-256

hash function to generate identifiers makes it possible to efficiently search for records in the register without the need to decrypt them beforehand. Thus, the Ethereum implementation demonstrates the viability of the proposed protocol for public networks, ensuring a balance between the transparency of the blockchain and the confidentiality of user information while maintaining economic efficiency within the limits of EVM gas.

3.3. Implementation on the Hyperledger Fabric Platform

To implement the protocol in the corporate segment, the Hyperledger Fabric platform version 2.5 was selected, which provides the necessary flexibility in configuring cryptographic modules and high throughput for the transfer of voluminous post-quantum artifacts. The network architecture is deployed in a virtualized WSL2 environment based on Ubuntu 24.04 using Docker Desktop, which allows you to simulate a distributed structure of two independent organizations. Each organization manages its own certificate authority and peering nodes, and CouchDB is used as the status database. CouchDB selection is critically important for this protocol, as the query mechanism allows efficient search and filtering of records by subject identifier hashes without the need to sequentially iterate through all the blocks of the register.

The integration of post-quantum cryptography in this implementation is carried out through a specialized software module based on the noble-post-quantum library, which provides support for the current NIST standards in 2024. The system supports a hybrid mode of operation, allowing for dynamic switching between classical cryptographic schemes based on elliptical curves and post-quantum algorithms based on modular lattices. In the post-quantum mode, instead of the standard Diffie–Hellman key exchange, the ML-KEM-768 key encapsulation mechanism is used, which provides secure generation of a symmetric secret for AES-256-GCM encryption. The ML-DSA-65 digital signature algorithm is used to authenticate each entry in the registry, which guarantees the quantum stability of the entire data ownership chain [29–34].

The software logic for the data management is concentrated in the ConfidentialDataContract smart contract, written in the Node language.js using the Fabric Contract API. The contract includes twelve functions covering the full life cycle of personal data from the moment of initial storage to validation by third-party verifiers. One of the key features of the Fabric implementation is the ability to store complete post-quantum signatures of 3.3 kilobytes directly in the World State. Unlike public networks, where such a volume of data would lead to a prohibitively high transaction cost, the Hyperledger Fabric corporate infrastructure allows processing such packets without degrading the economic model of the system, limited only by the requirements for disk space and bandwidth of internal communication channels.

For the experiment, let's take the same text with a volume of 30 kb (30,720 characters).

Data entry into the Hyperledger Fabric blockchain registry is presented in Figures 8–10.

In Figure 8, you can see the total amount of transmission of one message to another user using classical cryptography.

The result of transmitting a similar text using post-quantum algorithms is shown in Figure 11.

As we can see, the volume of transactions increased by 5 KB.

The client side interacts with the blockchain network through the Fabric Gateway API using gRPC connections. The Electron-based client application coordinates the process of executing the four-step protocol, providing automatic subscription to smart contract events, such as receiving a new access request or performing data reencryption. The key management module provides isolated storage of binary files with post-quantum key pairs, ensuring that secret parameters are never transmitted to the nodes of the blockchain.

Thus, its implementation for Hyperledger Fabric demonstrates the practical achievability of quantum stability in closed systems, where the requirements for personal data security prevail over the limitations on the amount of information stored.

The screenshot shows the 'Confidential Data Storage — Hyperledger Fabric' application. The top navigation bar includes 'Send Data', 'Request Data', 'Received Data', and 'Statistics'. The 'Send Data' tab is active. The form is titled 'Send Data to Blockchain (Step 1)'. It includes a 'Режим: Классическая (ECDH + ECDSA)' dropdown, a 'Шифрование: ECDH secp256k1 + AES-256-GCM | Подпись: ECDSA secp256k1 | Хэш: SHA-256' text, and a 'Subject Data (DataC):' text area containing Lorem ipsum text. Below this is a 'Subject Identifier (IdC):' input field with 'user1' and an 'Access for users:' input field with 'userB'. A 'Comma-separated, for example: userB' hint is present. A blue 'Encrypt and Send to Blockchain' button is at the bottom. On the right, a 'Submission Result' box states 'Result will appear after data submission'.

Figure 8. Send Data to Storage (Hyperledger Fabric).

The screenshot shows the 'Confidential Data Storage — Hyperledger Fabric' application. The top navigation bar includes 'Send Data', 'Request Data', 'Received Data', and 'Statistics'. The 'Received Data' tab is active. The form is titled 'Received Data (Step 4)'. It includes a 'Запрос: req_51a375e0' text, a 'Классическая' dropdown, and a 'Верифицировано' button. Below this is a 'Источник: Org1MSP | Время: 2026-03-25T23:16:59.000Z | Data ID: data_513d3b19' text. A 'Расшифрованные данные:' section contains a large text area with Lorem ipsum text. A 'Refresh' button is in the top right corner.

Figure 9. Received Data (Hyperledger Fabric).

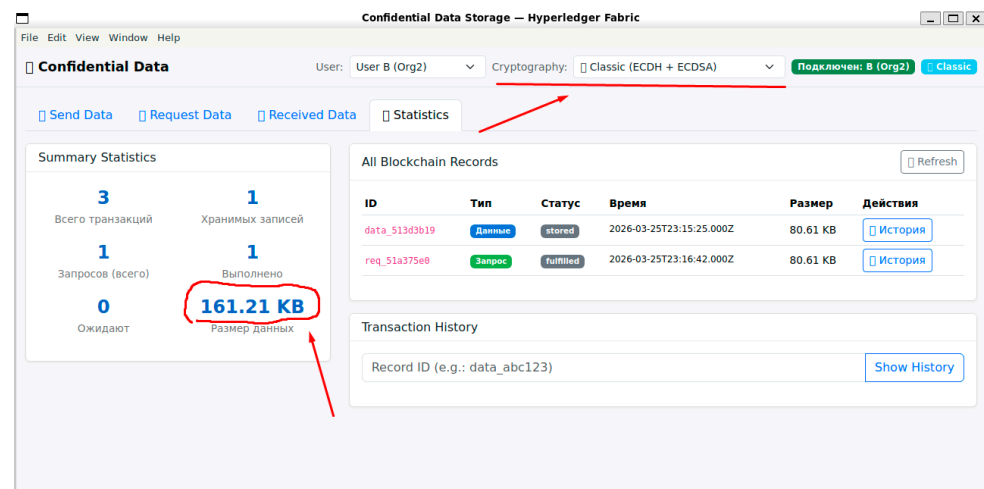


Figure 10. Registry Transaction Statistics (Hyperledger Fabric, ECDH + ECDSA).

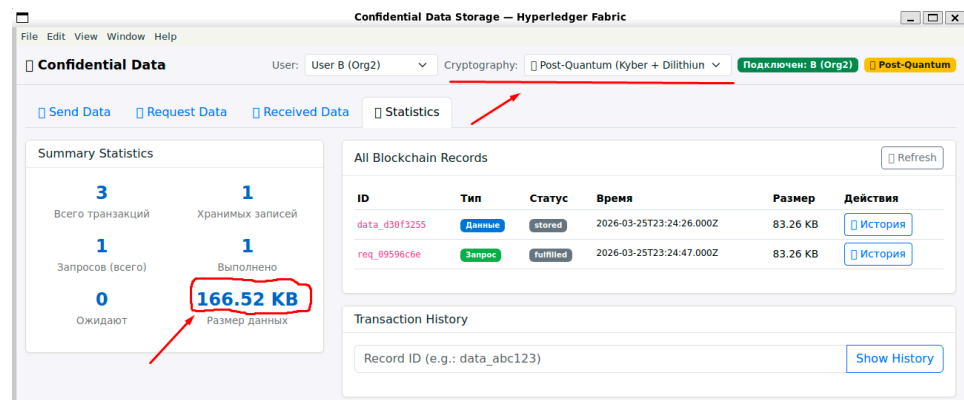


Figure 11. Registry Transaction Statistics (Hyperledger Fabric, Kyber + Dilithium).

However, a critical bottleneck is the unprecedented growth in the volume of cryptographic data transmitted and stored in the registry. The classic ECDSA signature takes only 64 bytes, while the ML-DSA-65 signature requires 3293 bytes, which is more than a 50-fold increase in the transaction size. Similarly, public keys in post-quantum schemes become 80 times larger than their classical counterparts. In public blockchain networks such as Ethereum, where the cost of each operation (Gas) directly depends on the volume of bytes transferred (calldata), such an increase in data volume leads to a proportional increase in transaction costs. This makes direct storage of post-quantum signatures in the main Layer 1 network economically inaccessible to ordinary users and mass applications.

The structural consequences for public networks also include a sharp drop in bandwidth. While maintaining the current gas limits per block, the transition to post-quantum standards will lead to a 90–95% reduction in the number of transactions per block, which will cause a constant overload of the mempool and an exponential increase in priority processing fees. Additionally, the problem of registry State Bloat arises since quantum-stable accounts require almost 60 times more disk space for storage in the World State, which accelerates network centralization due to increased system requirements for full nodes.

Conversely, in corporate and private networks such as Hyperledger Fabric, the implementation of PQC is technically justified and feasible. The lack of a public gas market allows organizations to scale their internal storage infrastructure and communication channels to absorb increased overhead costs without compromising business logic. Thus, a comparative analysis confirms that in the current conditions, PQC protocols are optimal for closed

systems, while public platforms require the use of signature aggregation technologies or second-level solutions (Layer 2) to maintain stability.

3.4. Comparison of Developed Protocols with Existing Ones

To clearly highlight the novel contributions and architectural advantages of our proposed framework, we compare our protocols (Protocols 1–4) with widely established cryptographic techniques and decentralized storage strategies. Table 13 summarizes this comparative analysis across five key dimensions requested for robust blockchain-based data sharing.

Table 13. Architectural and Cryptographic Comparison of the Proposed Protocols vs. Existing Techniques.

Method/Protocol	Primary Security Goals	On-Chain Overhead	Post-Quantum Resistance	Dynamic Authorization Support
ECIES	Asymmetric encryption and key encapsulation for a static recipient.	High (if full ciphertexts or large public keys are published directly to the ledger).	No	No (Static encryption tied to a specific public key).
ECDH	Interactive symmetric session key agreement.	High (requires explicit key exchange steps).	No	No (handshake must be repeated per new user).
Proxy Re-Encryption (PRE)	Ciphertext delegation to new recipients without decryption.	Medium (smart contracts manage re-encryption tokens).	No (typically uses classical pairings).	Yes (via uploading new proxy re-encryption keys).
On-Chain Access Control (AC)	Permission enforcement and public audit logging.	Medium to High (gas spent on executing complex ACL rules).	No (depends on ledger signature scheme).	Yes (via smart contract state and flag modifications).
Off-Chain Encrypted Storage	Scalable storage for bulk data, reducing ledger bloat.	Low (only data hashes or URIs/metadata anchored on-chain).	Depends on inner cipher (e.g., AES-256 is PQ-safe).	No (must be handled externally via separate KMS/PRE).
Hybrid Post-Quantum Encryption (PQE)	Long-term data confidentiality against quantum attacks	Very High (massive payload size due to PQ keys/signatures).	Yes	No (inherits static limitations of traditional asymmetric ciphers).
Proposed Protocol 1 (Secure asymmetric data exchange)	Decentralized, multi-party verified asymmetric data routing with strict identity-to-address binding.	Medium (stores data hash H^* , identity hash H_1 , and signature S_1).	Depends on inner cipher	Yes
Proposed Protocol 2 (validity verification)	Validity for third-party auditors (User D).	Medium (minimal data footprint; execution handled via ephemeral tokens).	Depends on inner cipher	Yes
Proposed Protocol 3 (Masked public-key transmission)	Secure, anonymous peer-to-peer exchange of public keys over a ledger without any trusted CA.	Medium (requires 3 sequential rounds of state submissions to the blockchain)	Depends on inner cipher	Yes
Proposed Protocol 4 (Shared-key-based data exchange)	Ultra-lightweight and fast data sharing using pre-established symmetric secrets.	Low (only metadata hashes and symmetric ciphertext M_1 are processed)	Depends on inner cipher	Yes

The integration of Post-Quantum Cryptography (PQC) into decentralized ledgers introduces a severe technical trade-off: while algorithms like ML-DSA (CRYSTALS-Dilithium) or Falcon offer robust resistance against Shor’s algorithm, their public keys and signatures

are orders of magnitude larger than classical 32-byte ECDSA keys. For instance, an ML-DSA-6523 signature requires over 3300 bytes. If directly injected into a public ledger's state, this data inflation triggers exponential gas fee validation hikes and severe ledger storage bloat. To guarantee the practical applicability of our four protocols during the quantum transition, we propose concrete deployment strategies, migration pathways, and an architectural impact analysis.

Practical Deployment and Storage Mitigation Strategies. To prevent post-quantum signatures and keys from paralyzing public blockchain state storage, we outline three key architectural strategies:

Off-Chain Data Availability (DA) Layers: Instead of storing heavy post-quantum signatures (S_1, S_2) directly on-chain within the smart contract state, the cryptographic payloads are offloaded to specialized Data Availability networks (e.g., Celestia or EigenDA). The primary public ledger anchors only a lightweight 32-byte cryptographic root hash (commitment) of the signatures. This maintains full verification accountability while reducing on-chain storage overhead by up to 95%.

Layer-2 Optimistic and ZK-Rollups: Executing Protocols 1–3 directly on Layer-1 (L1) public networks like Ethereum Mainnet during the PQC era would be cost-prohibitive. By executing the multi-pass exchanges of Protocol 3 or the validation tokens of Protocol 2 on Layer-2 (L2) rollups, thousands of heavy PQC transactions are compressed and bundled into a single succinct proof (e.g., STARK/SNARK) before being anchored to L1, slashing gas overhead.

Ephemeral State Pruning: Protocols 2 and 3 rely on interactive, time-locked, or multi-pass stages that become obsolete once the transaction finality window closes. Smart contracts should be engineered with automated data pruning triggers, clearing intermediate masked parameters (X, Y, Z) from the contract's active storage slot once the public keys or verifications are successfully derived, converting persistent storage costs into cheaper ephemeral execution costs.

Phased Migration Pathways from Classical Cryptography. Transitioning an active, multi-user blockchain infrastructure to a post-quantum standard cannot occur overnight. We propose a three-phased migration pathway.

Phase 1. Hybrid Dual-Signature Regime (Short-to-Medium Term). Accounts are assigned dual-key pairs containing both a classical ECDSA key and an emerging lattice-based ML-DSA key. Transactions must be signed by both primitives. If an adversary compromises the classical layer using a prototype quantum computer, the lattice layer ensures data confidentiality and identity integrity.

Phase 2. Post-Quantum Address Derivation. Current blockchain networks derive 20-byte user addresses (IdA, IdB) from the Keccak-256 hash of classical public keys. During migration, the address derivation formula must be updated to abstract the underlying key type, using a prefix byte (e.g., 0x01 for ECC, 0x02 for ML-KEM/Lattice), allowing smart contracts in Protocol 1 and 2 to transparently route data based on the recipient's exact cryptographic generation.

Phase 3: Hard-Fork Deprecation. Once post-quantum hardware-accelerated verification signatures achieve broad standardization, classical single-key accounts are marked as read-only or deprecated via network-level hard forks, forcing users to migrate funds and data registries to pure PQC structures.

Quantitative Impact on Public Blockchain Infrastructure. The definitive architectural advantage of our framework lies in its structural diversity, particularly the utilization of Protocol 4. Traditional data sharing models force the persistent execution of asymmetric public-key operations. In a post-quantum public ledger, every single verification step would require nodes to fetch and compute over kilobytes of lattice matrices, severely

degrading network throughput (Transactions Per Second, TPS). By implementing Protocol 4 for repetitive bulk data exchanges, the system completely bypasses this post-quantum bottleneck. Once the initial PQC peer identity is established, the ongoing on-chain execution relies strictly on symmetric AES-256-GCM and hash verifications ($H1$, H^*). Since symmetric primitives are naturally resilient to quantum attacks (requiring only a baseline 256-bit key length to withstand Grover's algorithm) and maintain microscopic byte sizes, Protocol 4 insulates the host public blockchain infrastructure from state bloat and computational exhaustion, making the entire framework highly viable for real-world, high-throughput enterprise deployments.

3.5. Experimental Evaluation of the Overhead Costs of the Developed Protocols When Using Post-Quantum Algorithms

The integration of post-quantum cryptographic (PQC) primitives significantly impacts data storage requirements due to the larger key and signature sizes compared to classical curves (e.g., ECDSA). Table 14 summarizes the storage overhead for the developed protocols.

We consider Protocols 1, 2, and 4, using lattice KEM algorithms (ML-KEM/CRYSTALS-Kyber). This is the de facto standard approved by NIST. Instead of asymmetric ECIES encryption, user A uses a post-quantum key encapsulation mechanism (KEM) to agree on a symmetric key with user D and then encrypts the data with a symmetric cipher, such as AES-256-GCM. For Protocol 3, this use case is unsuitable, since most NIST-approved post-quantum algorithms (such as lattice ML-KEM/Kyber or ML-DSA/Dilithium) do not have the commutativity property. The only mature post-quantum algorithm family that naturally has commutativity is Commutative Supersingular Isogeny Diffie–Hellman (CSIDH). The group of classes of acting endomorphisms behaves exactly like classical exponentiation. Instead of the numbers k_A and k_B , users generate random ideals (step vectors along the isogeny graph), and instead of exponentiation, they perform isogeny calculations.

Table 14. Storage overhead per protocol/transaction.

Protocol	Basic Primitives	Key/Data Size (Bytes)	Total Transaction Payload Size (Bytes)
Protocol 1	ML-KEM-768	Public key: 1184 Ciphertext: 1088	~2350
Protocol 2	ML-KEM-768	Public key: 1184 Ciphertext: 1088	~512
Protocol 3	CSIDH	One transferred value: 64 bytes	for all steps of the protocol
Protocol 4	ML-KEM-768	Public key: 1184 Ciphertext: 1088	~2350

For deployment on the Ethereum network, gas consumption was estimated using the Solidity compiler (solc 0.8.20) with optimizations enabled (200 cycles) and is presented in Table 15.

Table 15. Gas consumption for Ethereum smart contracts.

Smart Contract Function/Operation	Gas Used (Classic ECIES/ECDSA)
Deployment	1,250,000
Protocol 1	45,000
Protocol 2	21,000
Protocol 3	22,000

Transaction throughput (transactions per second, TPS) and transaction latency (in seconds) were tested under various load levels. For Hyperledger Fabric 2.5, the Hyperledger Caliper tool was used to generate loads ranging from 100 to 2000 concurrent transaction requests. For the Ethereum environment, the private Ethereum Proof-of-Authority (PoA) testnet was used to eliminate the influence of external network congestion factors.

In Hyperledger Fabric, throughput grew linearly up to a saturation point of 1250 TPS for Protocol 1, due to the efficient Raft consensus and state management via CouchDB. For Ethereum, throughput was limited by the block gas limit (30 million gas), peaking at 120 TPS.

Under baseline load (<500 concurrent requests), Hyperledger Fabric demonstrated a low average latency of 0.15 s. Under peak load (2000 requests), latency increased to 1.2 s. On Ethereum, average latency remained tied to block generation time and averaged 2.3 s under normal conditions.

Protocol 3 introduces overhead due to the multi-round message exchange. The network traffic for one complete key exchange sequence is approximately 3.6 KB of data transferred over the network. This volume is well within the bandwidth of modern corporate and public networks.

Experimental results show that although the transition to post-quantum cryptography introduces significant overhead (especially in gas consumption and storage requirements), the system maintains acceptable latency and high throughput in private deployment architectures (Hyperledger Fabric), making it highly effective for enterprise secure data exchange.

4. Conclusions

An analysis of the current state of cryptographic protocols confirms that the quantum threat to blockchain systems has ceased to be theoretical and requires immediate protection planning. The “Harvest Now, Decrypt Later” strategy makes the implementation of post-quantum algorithms critically important today, since any confidential data encrypted by classical methods and placed in an immutable registry now is virtually guaranteed to be disclosed with the advent of quantum computers, expected in the period 2030 ± 3 years. Today, the technological base is fully ready for the transition, as the National Institute of Standards and Technology of the USA (NIST) officially finalized key standards such as ML-KEM for key encapsulation and ML-DSA for digital signatures in August 2024. The practical implementation of these algorithms through modern libraries, such as noble-post-quantum, demonstrates high computational efficiency and verification speed, which in some cases surpasses the performance of the classical ECDSA. Nevertheless, the unprecedented volume of post-quantum artifacts remains the main system obstacle: a 50-fold increase in the signature size (from 65 bytes to 3309 bytes for ML-DSA-65) leads to a sharp increase in the registry state (state bloat) and a critical decrease in the bandwidth of public networks. While corporate platforms like Hyperledger Fabric can successfully absorb such overhead costs by scaling internal infrastructure, Layer 1 public networks will require the use of account abstraction mechanisms (ERC-4337), hybrid schemes, and second-level solutions to maintain economic efficiency [34–46].

This paper examines the problem of ensuring the quantum stability of cryptographic protocols in blockchain systems in the context of protecting personal data from the “Harvest Now, Decrypt Later” strategy. A family of protocols for data exchange with the division of users into access categories is proposed and implemented on two contrasting platforms: public (Ethereum) and corporate (Hyperledger Fabric 2.5). The integrated elliptic curve encryption (ECIES) scheme with Keccak-256 hashing was used for Ethereum, while the latest NIST—ML-KEM and ML-DSA post-quantum standards based on lattice cryptography were integrated for Hyperledger Fabric.

A comparative analysis performed has shown that post-quantum algorithms provide high computing and verification speeds, but they are accompanied by a significant increase in the volume of cryptographic artifacts: the signature size increases by more than 50 times and public keys by 80 times relative to classical analogs. It makes the direct implementation

of post-quantum cryptography in public Layer 1 networks economically impractical due to the proportional increase in transaction costs and reduced bandwidth. At the same time, in closed and corporate environments, such as Hyperledger Fabric, these overhead costs can be absorbed by a scalable infrastructure, which makes the implementation of PQC technically justified and feasible.

Based on the results obtained, the recommendations are to use a hybrid security model combining classical and post-quantum algorithms, as well as to apply second-level (Layer 2) solutions and signature aggregation mechanisms for public blockchain networks. The proposed architecture and protocols create the basis for a phased migration to quantum-stable blockchain systems capable of ensuring long-term confidentiality of personal data in the context of developing quantum computing technologies.

Our future research plans include strengthening the experimental phase for various blockchain platforms using different types of encryption algorithms to determine the optimal balance between processing speed and the amount of data stored in the blockchain, depending on the platform used.

Author Contributions: Conceptualization, K.R., E.I. and S.P.; methodology, K.R., E.I. and S.P.; software, K.R., E.I. and A.P.; validation, K.R., E.I., S.P., A.P. and A.N.; formal analysis, K.R., E.I., S.P. and A.N.; investigation, K.R., E.I., S.P., A.P. and A.N.; resources, S.P.; data curation, K.R., E.I., S.P., A.P. and A.N.; writing—original draft preparation, K.R. and E.I.; writing—review and editing, K.R., E.I., S.P., A.P. and A.N.; visualization, K.R., E.I. and A.P.; supervision, E.I.; project administration, S.P.; funding acquisition, S.P. All authors have read and agreed to the published version of the manuscript.

Funding: The results were obtained with the financial support of the project “Technologies for countering previously unknown quantum cyber threats”, implemented within the framework of the state program of the “Sirius” Federal Territory “Scientific and technological development of the “Sirius” Federal Territory (Agreement No. 23-03 dated 27 September 2024). The research was funded by the Russian Science Foundation (project No. 25-11-20037, <https://rscf.ru/project/25-11-20037/> (accessed on 6 July 2026)). The APC was funded by the Russian Science Foundation (project No. 25-11-20037, <https://rscf.ru/project/25-11-20037/> (accessed on 6 July 2026)).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
AIoMT	Artificial Intelligence of Medical Things
AML	Anti-Money Laundering
BIP	Bitcoin Improvement Proposals
CDHP	Computational Diffie–Hellman problem
CSIDH	Commutative Supersingular Isogeny Diffie–Hellman
dApps	Decentralized Applications
DHT	Distributed Hash Table
DLP	Discrete Logarithm Problem
ECDSA	Elliptic Curve Digital Signature Algorithm
ECIES	Elliptic Curve Integrated Encryption Scheme
EIP	Ethereum Improvement Proposals
EUFCMA	Existential Unforgeability under Chosen Message Attacks
EVM	Ethereum Virtual Machine

HKDF	HMAC-based Key Derivation Function
IND-CCA2	Indistinguishability under Adaptive Chosen-Ciphertext Attack
IoT	Internet of Things
IT	Information Technology
MitM	Man-in-the-Middle
ML-DSA	Module-Lattice-Based Digital Signature Algorithm
ML-KEM	Module-Lattice-Based Key Encapsulation Mechanism
PBFT	Practical Byzantine Fault Tolerance
PoA	Proof-of-Authority
PoS	Proof of Stake
PoW	Proof of Work
PRNG	Perfect Random Number Generator
PRS	Pseudo-Random Sequence
PUF	Physically Unclonable Functions
PQC	Post-Quantum Cryptography
TPS	Transactions Per Second
VANET	Vehicular Ad hoc Network
V2G	Vehicle-to-Grid

References

1. Post-Quantum Cryptography | CSRC—NIST Computer Security Resource Center. Available online: <https://csrc.nist.gov/project-s/post-quantum-cryptography> (accessed on 31 May 2026).
2. Zyskind, A.; Nathan, A.; Pentland, A. Decentralizing Privacy: Using Blockchain to Protect Personal Data. In Proceedings of the 2015 IEEE Security and Privacy Workshops, San Jose, CA, USA, 18–20 May 2015; pp. 180–184. [\[CrossRef\]](#)
3. Finck, M. Blockchain and the General Data Protection Regulation: Can Distributed Ledgers Be Squared with European Data Protection Law? European Union Blockchain Observatory & Forum 2019, 108p. Available online: [https://www.europarl.europa.eu/RegData/etudes/STUD/2019/634445/EPRS_STU\(2019\)634445_EN.pdf](https://www.europarl.europa.eu/RegData/etudes/STUD/2019/634445/EPRS_STU(2019)634445_EN.pdf) (accessed on 31 May 2026).
4. Dinh, T.T.A.; Liu, R.; Zhang, M.; Chen, G.; Ooi, B.C.; Wang, J. Untangling Blockchain: A Data Processing View of Blockchain Systems. *IEEE Trans. Knowl. Data Eng.* **2018**, *30*, 1366–1385. [\[CrossRef\]](#)
5. Kosba, A.; Miller, A.; Shi, E.; Wen, Z.; Papamanthou, C. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts. In Proceedings of the 2016 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 23–25 May 2016; pp. 839–858. [\[CrossRef\]](#)
6. Shuaib, M.; Hassan, N.H.; Usman, S.; Alam, S.; Sam, S.M.; Samy, G.A.-L.N. Effect of Quantum computing on Blockchain-based Electronic Health Record Systems. In Proceedings of the 2022 4th International Conference on Smart Sensors and Application (ICSSA), Kuala Lumpur, Malaysia, 26–28 July 2022; pp. 179–184. [\[CrossRef\]](#)
7. Quantum Computing and Blockchain Security: A Critical Assessment of Cryptographic Vulnerabilities and Post-Quantum Migration Strategies. Available online: https://mpira.ub.uni-muenchen.de/127528/1/MPRA_paper_127528.pdf (accessed on 31 May 2026).
8. “Harvest Now Decrypt Later”: Examining Post-Quantum Cryptography and the Data Privacy Risks for Distributed Ledger Networks—Federal Reserve Board. Available online: <https://www.federalreserve.gov/econres/feds/files/2025093pap.pdf> (accessed on 31 May 2026).
9. NIST Post-Quantum Cryptography Standardization—Wikipedia. Available online: https://en.wikipedia.org/wiki/NIST_Post-Quantum_Cryptography_Standardization (accessed on 31 May 2026).
10. Liang, Y.; Chen, J.; Bu, L.; Liu, T.; Wang, X.; Wang, B.; Zhang, G.; Sun, H. Fully Decentralized Authentication and Key Exchange Scheme for Data Sharing in AIoMT. *IEEE Trans. Ind. Inform.* **2026**, *1*–12. [\[CrossRef\]](#)
11. Liang, Y.; Wan, Z.; Yiming, C.; Zhao, Z.; Liu, G.; Liu, Z.; Sun, H.; Liu, Y. Vehicle-Side Lightweight Conditional Anonymous Double Authentication Framework with Password Robustness in VANETs. *IEEE Trans. Dependable Secure Comput.* **2026**, *1*–18. [\[CrossRef\]](#)
12. Liang, Y.; Sun, H.; Zhang, X.; Xie, Q.; Liu, G.; Liu, Z. Lightweight Multifactor Authentication and Key Agreement Scheme in Vehicle-to-Grid Networks. *IEEE Trans. Veh. Technol.* **2026**, *75*, 6582–6599. [\[CrossRef\]](#)
13. Liang, Y.; Liu, G.; Zhang, X.; Chen, Y.; Chen, J.; Xia, Y. Fully Anonymous Broadcast Signcryption for Secure Health Data Transmission in WBANs. *IEEE Trans. Mob. Comput.* **2026**, *25*, 1729–1743. [\[CrossRef\]](#)
14. Harvest Now, Decrypt Later (HNDL): The Quantum-Era Threat—Palo Alto Networks. Available online: <https://www.paloaltonetworks.com/cyberpedia/harvest-now-decrypt-later-hndl> (accessed on 31 May 2026).

15. Ali, S.; Wadho, S.A.; Talpur, K.R.; Talpur, B.A.; Alshudukhi, K.S.; Humayun, M.; Talpur, S.R.; Al Mamun, M.A.; Naseem, M.; Abro, A.; et al. Next-Generation Quantum Security: The Impact of Quantum Computing on Cybersecurity—Threats, Mitigations, and Solutions. *Comput. Electr. Eng.* **2025**, *128*, 110649. [CrossRef]
16. Kagai, F.; Branch, P.; But, J.; Allen, R. Harvest-Now, Decrypt-Later: A Temporal Cybersecurity Risk in the Quantum Transition. *Telecom* **2025**, *6*, 100. [CrossRef]
17. Harvest Now, Decrypt Later: A Time-Dependent Threat Model and Migration Framework for Post-Quantum Cryptography—ResearchGate. Available online: https://www.researchgate.net/publication/400298687_Harvest_Now_Decrypt_Later_A_Time-Dependent_Threat_Model_and_Migration_Framework_for_Post-Quantum_Cryptography (accessed on 31 May 2026).
18. “Harvest Now, Decrypt Later” Quantum Threat for Regulated Industries—SafeLogic. Available online: <https://www.safelogic.com/blog/harvest-now-decrypt-later-quantum-threat> (accessed on 31 May 2026).
19. Why Your Encrypted Data from 2019 Is Already Compromised: The Quantum Time Bomb. Available online: <https://gupta-deepak.com/why-your-encrypted-data-from-2019-is-already-compromised-the-quantum-time-bomb/> (accessed on 31 May 2026).
20. Quantum Threat—Citi. Available online: https://www.citigroup.com/rcs/citigpa/storage/public/Citi_Institute_Quantum_Threat.pdf (accessed on 31 May 2026).
21. Campbell, R. Hybrid Post-Quantum Signatures for Bitcoin and Ethereum: A Protocol-Level Integration Strategy. *J. Br. Blockchain Assos.* **2026**, *9*, 1–13. [CrossRef] [PubMed]
22. Chhetri, G.; Somvanshi, S.; Hebli, P.; Brotee, S.; Das, S. Post-Quantum Cryptography and Quantum-Safe Security: A Comprehensive Survey. *arXiv* **2025**, arXiv:2510.10436v1. Available online: <https://arxiv.org/html/2510.10436v1> (accessed on 31 May 2026).
23. What Is Post-Quantum Cryptography (PQC)? A Complete Guide—Palo Alto Networks. Available online: <https://www.paloaltonetworks.com/cyberpedia/what-is-post-quantum-cryptography-pqc> (accessed on 31 May 2026).
24. NIST FIPS 203, 204, 205 Finalized | PQC Algorithms—Cloud Security Alliance (CSA). Available online: <https://cloudsecurityalliance.org/blog/2024/08/15/nist-fips-203-204-and-205-finalized-an-important-step-towards-a-quantum-safe-future> (accessed on 31 May 2026).
25. Lattice-Based Signature—Cryptology ePrint Archive. Available online: <https://eprint.iacr.org/search?q=lattice-based%20signatures> (accessed on 31 May 2026).
26. Mallick, T.; Nita-Rotaru, C.; Kundu, A.; Kompella, R. Classifying Implementations of Cryptographic Primitives and Protocols that Use Post-Quantum Algorithms. *arXiv* **2026**, arXiv:2503.17830v4. Available online: <https://arxiv.org/html/2503.17830v4> (accessed on 31 May 2026).
27. Ashok, G.A.; Sakharan, C.M.; Ashok, J.S.; GK, G. Blockchain Based Proxy Re-Encryption for Secure Data Sharing—ResearchGate. *Int. J. Adv. Res. Sci. Commun. Technol.* **2024**, *4*, 236–241. [CrossRef]
28. The Quantum Event Horizon: Cryptographic Vulnerabilities in the Ethereum Network. Available online: <https://dev.to/extropy/the-quantum-event-horizon-cryptographic-vulnerabilities-in-the-ethereum-network-23a9> (accessed on 31 May 2026).
29. Melo, C.; Gonçalves, G.; Silva, F.A.; Soares, A. Performance Modeling and Evaluation of Hyperledger Fabric: An Analysis Based on Transaction Flow and Endorsement Policies. *arXiv* **2025**, arXiv:2502.08755. Available online: <https://arxiv.org/pdf/2502.08755> (accessed on 31 May 2026).
30. GitHub—Paulmillr/Noble-Post-Quantum: Auditable & Minimal JS Implementation of Public-Key Post-Quantum Cryptography. Available online: <https://github.com/paulmillr/noble-post-quantum> (accessed on 31 May 2026).
31. FIPS 204. In Our Previous Exploration of ML-KEM. . . | by kcl17 | Medium. Available online: <https://medium.com/@kcl17/ml-dsa-fips-204-dd151ace3493> (accessed on 31 May 2026).
32. Gonçalves, G.; Silva, F.; Soares, A. A Comprehensive Hyperledger Fabric Performance Evaluation based on Resources Capacity Planning. *arXiv* **2025**, arXiv:2502.10509v1. Available online: <https://arxiv.org/html/2502.10509v1> (accessed on 31 May 2026).
33. Towards Quantum-Safe Smart Contracts on Hyperledger Fabric | TU Delft Repository. Available online: <https://resolver.tudelft.nl/uuid:cec0b1bf-fc85-4d58-a4c0-6a81e184a3a7> (accessed on 31 May 2026).
34. Schemitt, A.G.; da Silva, H.F.; Lunardi, R.C.; Kreutz, D.; Mansilha, R.B.; Zorzo, A.F. Assessing the Impact of Post-Quantum Digital Signature Algorithms on Blockchains. *arXiv* **2025**, arXiv:2510.09271v1. Available online: <https://arxiv.org/html/2510.09271v1> (accessed on 31 May 2026).
35. Chhetri, R. Benchmarking NIST-Standardised ML-KEM and ML-DSA on ARM Cortex-M0+: Performance, Memory, and Energy on the RP2040. *arXiv* **2026**, arXiv:2603.19340v1. Available online: <https://arxiv.org/html/2603.19340v1> (accessed on 31 May 2026).
36. ML-DSA-44 in Rust—ASecuritySite.com. Available online: https://asecuritysite.com/rust/rust_mldsa44 (accessed on 31 May 2026).
37. Benchmarking ML-DSA Signature Generation: Understanding Rejection Sampling Performance—Among Bytes. Available online: <https://www.amongbytes.com/posts/20260223-mldsa-benchmarking/20260223-mldsa-benchmarking.html> (accessed on 31 May 2026).
38. Taherdoost, H. A Comprehensive Review of Quantum-Resistant Architectures for Blockchain Security. *Sci* **2026**, *8*, 47. [CrossRef]
39. Announcing Quantum-Resistant Security on Streamr. Available online: <https://blog.streamr.network/announcing-quantum-resistant-security-on-streamr/> (accessed on 31 May 2026).

40. Post-Quantum-Cryptography-GitHub Topics. Available online: <https://github.com/topics/post-quantum-cryptography?o=desc&s=forks> (accessed on 31 May 2026).
41. Project Eleven. Bulletin #15 | Ethereum's Quantum Attack Vectors & ECDLP Challeng. Available online: <https://blog.project-eleven.com/posts/bulletin-15--ethereums-quantum-attack-vectors--ecdip-challeng> (accessed on 31 May 2026).
42. Khodaiemehr, H.; Bagheri, K.; Feng, C. Navigating the quantum computing threat landscape for blockchains: A comprehensive survey. *Comput. Sci. Rev.* **2026**, *59*, 100846. [CrossRef]
43. Quantum Disruption: An SOK of How Post-Quantum Attackers Reshape Blockchain Security and Performance—ResearchGate. Available online: https://www.researchgate.net/publication/398720912_Quantum_Disruption_An_SOK_of_How_Post-Quantum_Attackers_Reshape_Blockchain_Security_and_Performance (accessed on 31 May 2026).
44. Transitioning to Quantum-Safe Cryptography on IBM Z. Available online: <https://www.redbooks.ibm.com/redpieces/pdfs/sg24-8525.pdf> (accessed on 31 May 2026).
45. Jena, J. The Quantum Security Deadline: Building Crypto-Agility Against Harvest Now, Decrypt Later' Threats. *Eur. J. Comput. Sci. Inf. Technol.* **2025**, *13*, 35–53. Available online: <https://ejournals.org/ejcsit/wp-content/uploads/sites/21/2025/10/The-Quantum-Security-Deadline.pdf> (accessed on 31 May 2026). [CrossRef]
46. Dziechciarz, D.; Niemiec, M. Efficiency Analysis of NIST-Standardized Post-Quantum Cryptographic Algorithms for Digital Signatures in Various Environments. *Electronics* **2025**, *14*, 70. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.