# Anonymous Kerberos Authentication

Carol Chen
MIT
carol120@mit.edu

Evan Hong
MIT
ehong129@mit.edu

Luis Turino
MIT
turino14@mit.edu

***Abstract*—This paper proposes a novel enhancement to the Kerberos authentication protocol by incorporating zero-knowledge arguments of knowledge (zk-SNARK), specifically the Groth16 scheme, to address privacy concerns while maintaining strong security guarantees. In the Kerberos model, the Key Distribution Center (KDC) is a central point of trust, responsible for both authenticating users and safeguarding their passwords. However, this position of trust allows the KDC to keep track of logs of user activities and, if compromised, could result in the leakage of much sensitive data and the potential exposure of user accounts. By integrating Groth16, we enable authentication without the KDC ever learning or storing the user's password, or inferring private information from user-to-service interactions. The proposed modification ensures that even in the case of a compromised KDC, user privacy and system integrity are preserved. This paper explores the technical implementation of Groth16 within the Kerberos framework and demonstrates how zero-knowledge arguments of knowledge can enhance privacy while being efficient and maintaining the robustness of the original protocol.**

## I. Introduction

User authentication is a fundamental aspect of any service. Whether the goal is to restrict access to a select group or to enforce subscription-based usage, authentication mechanisms are essential components of system design.

A wide range of authentication schemes have been integrated into services ranging from online banking to email systems. Among these, a prevalent method is the username-and-password model. In this scheme, the client sends their username to the server, typically in plaintext, which the server uses to locate an associated encrypted key. This key can only be decrypted using the correct password. Authentication is successful only when both the username and password are valid. A user who possesses only one of these, either a valid username or password, will either be unable to retrieve the encrypted key or unable to decrypt it.

While authentication schemes are designed to protect servers from unauthorized clients, there are cases where legitimate users may wish to preserve their privacy during the authentication process. This introduces a more nuanced challenge: how can a system verify a user's identity without learning anything specific about that user? Although this idea may initially seem counterintuitive, several established solutions address this very concern. These solutions fall under the category of zero-knowledge arguments of identity, a family of cryptographic protocols that enable authentication without revealing any underlying private information.

One such scheme is Groth16, a variant of zero-knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARKs), which relies on the computational hardness of Quadratic Arithmetic Programs (QAPs) as its underlying foundation. In this paper, we explore the applicability of the Groth16 authentication scheme within the context of Kerberos, a ticket-based authentication protocol developed and maintained by MIT. Our motivation stems from the significant trust placed in the Key Distribution Center (KDC) of the traditional Kerberos model. The KDC is central not only to mediating user-to-service communications, but also to safeguarding users' plaintext passwords. Since every time a user needs to use a service, it must tell the KDC the service name, the KDC can keep track of a detailed log of services the users utilize. Moreover, a compromised KDC can lead to both the leakage of sensitive information and widespread account leakage. To mitigate these risks, we propose a modified authentication scheme that integrates zero-knowledge arguments in such a way that even a compromised KDC cannot gain access to user

credentials or infer meaningful information about user-to-service interactions.

## II. Kerberos Overview

### A. The Three Components: Client, KDC, Service

For an overview of the Kerberos protocol, we refer to *The Kerberos Network Authentication Service (V5)* by Neuman et al. [1].

*1) Client:* The first component in the Kerberos authentication scheme is the client, which represents the user attempting to authenticate with the Key Distribution Center (KDC) in order to access a desired service. The client possesses a user key, derived from the user's password, as well as their username, both of which are used during the authentication process.

*2) Key Distribution Center:* The KDC is the central authority responsible for initial authentication in the Kerberos protocol. The KDC is composed of the Authenticating Server (AS) and the Ticket Granting Service (TGS). The AS verifies a user's identity and issues a ticket-granting-ticket (TGT). The user utilizes the TGT to obtain service tickets from the TGS, which allow authenticated users to securely access services without repeatedly proving their credentials. The KDC maintains a copy of all user and service keys, and generates session keys.

*3) Service:* The service is the resource the client ultimately wishes to access. Access to the service is granted only after successful authentication through the KDC. Each service maintains its own service key, which is used to validate the tickets issued by the KDC and establish secure communication with the client.

### B. Protocol

See Figure 1 for a diagram of the original Kerberos protocol.

1) The client initiates the authentication process by sending their username in plaintext to the Authentication Server component of the KDC.
2) The KDC then responds with two items: a Ticket Granting Ticket (TGT), which is encrypted using the Ticket Granting Service (TGS) key, and a session key (denoted as $A$), which is encrypted using the client's unique user key. This session key includes information such as the requested service and the client's identity, enabling secure communication in the next stage of authentication.
3) The client decrypts the message using their user key to recover the session key $A$. Using this session key, the client then sends a request to the TGS, which includes the TGT, the user's username encrypted with session key $A$, and the name of the requested service in plaintext.
4) The TGS component of the KDC decrypts the TGT using its copy of the TGS key, thereby recovering the session key $A$. With this session key, the TGS can then decrypt the user's encrypted username, verifying the client's identity and ensuring the request is legitimate.
5) The KDC then responds by issuing a service ticket, which is encrypted using the service's key and contains a new session key $B$. This ticket is intended for the service the client wishes to access. In addition, the KDC sends the session key $B$ to the client, encrypted with the previously established session key $A$, allowing the client to decrypt and use it without exposing it to the service.
6) Since the client previously received session key $A$ from the KDC, they can use it to decrypt the message and recover session key $B$. With this key, the client encrypts their username using session key $B$ and sends it to the service, along with the service ticket obtained from the KDC.
7) The service, using its own copy of the service key, decrypts and processes the service ticket. To complete the authentication process, it then sends a confirmation message back to the client, encrypted with session key $B$.

### C. Lack of Zero Knowledge and Security Concerns

As demonstrated by the protocol the KDC holds the user keys of all users. With each user's username being public as it is tied to their emails, if the KDC is ever compromised and user passwords are leaked, all accounts would be compromised.

Another security concern is the lack of zero knowledge. When executing the protocol, the client explicitly sends over his identity in plaintext for the KDC to then retrieve the service key wanted by the

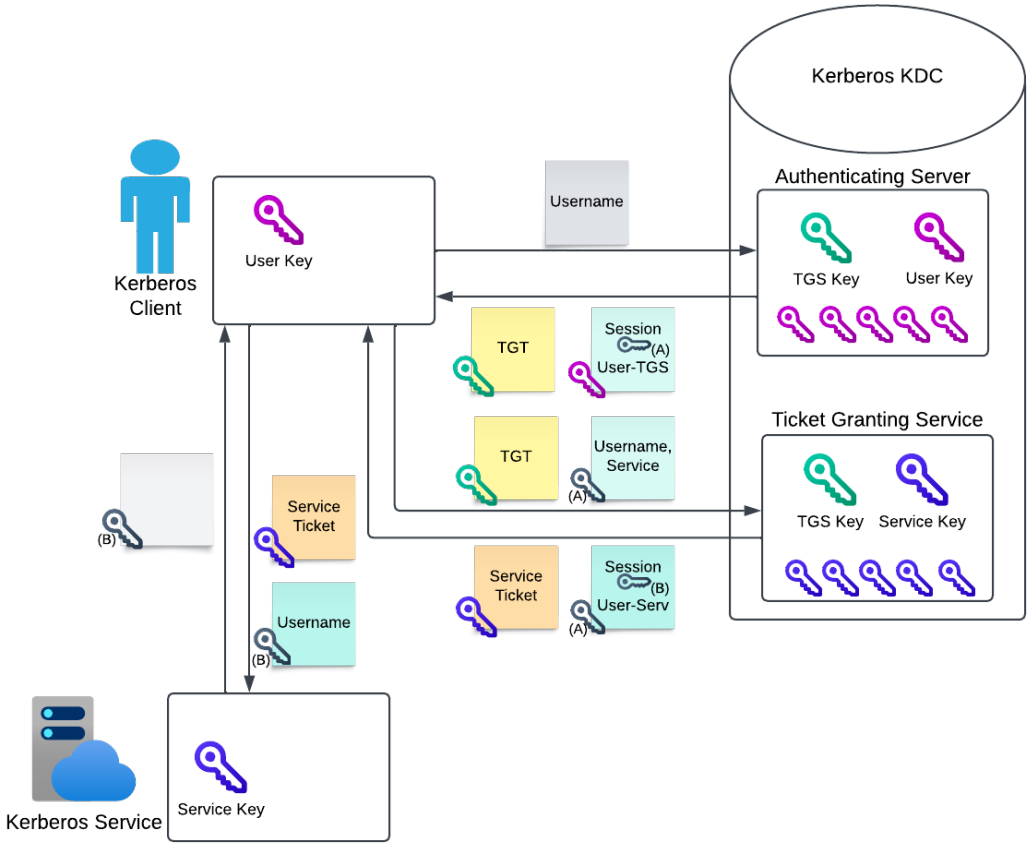


Fig. 1. Diagram detailing Kerberos authentication and service access protocols

client. Although this scenario isn't as bad as the first, the KDC will have the capability of keeping track of client activity across services.

## III. Zero Knowledge Design

### A. *zk-SNARK's Requirements*

In a zero knowledge scheme there exists a prover and a verifier. The prover holds knowledge to a key that he wishes to prove to the verifier without explicitly telling the verifier the value of his key. The verifier is to evaluate the arguments set forth by the prover and either accept or reject. To ensure both efficiency and privacy, zk-SNARKs must satisfy four essential cryptographic properties: zero-knowledge, succinctness, non-interactiveness, and argument of knowledge. These properties collectively enable a prover to convince a verifier of a statement's truth without revealing any additional information, while maintaining minimal communication overhead and computational burden. In the context of authentication systems like Kerberos, these guarantees are crucial for preserving client privacy and enabling scalable verification.

1) **Zero Knowledge:** In a zero-knowledge proof, the prover possesses a solution to an NP problem and should be able to convince a verifier of this knowledge without revealing the solution itself. Moreover, the interaction between the prover and the verifier can be simulated entirely by the verifier, ensuring that no knowledge beyond the validity of the claim is transferred.
2) **Succinctness:** A key property of zk-SNARKs is succinctness, meaning that the size of the proof (or argument) grows sublinearly with respect to the size of the solution, also called the witness. In the case of Groth16, the proof size is constant and completely independent

of the size of the witness, making it particularly well-suited for efficient verification in constrained environments.

3) **Non-interactiveness:** The proof process is non-interactive, meaning it does not require multiple rounds of communication between the prover and the verifier. Instead, the prover generates a single, self-contained argument, which the verifier can independently accept or reject.
4) **Argument of Knowledge:** Arguments of Knowledge assume a weaker, polynomial bounded prover, in contrast to traditional Proofs of Knowledge which assume a powerful, computationally unbounded prover.

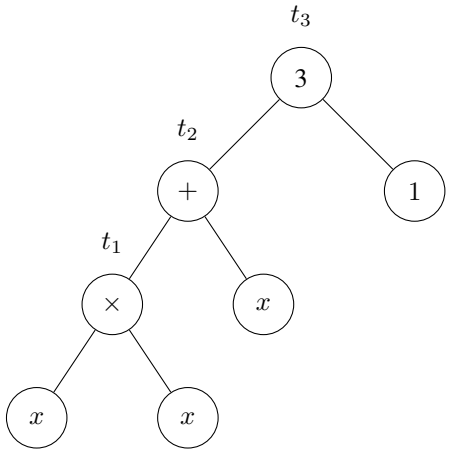


Fig. 2. Arithmetic Circuit tree with gates $t_1, t_2, t_3$ representing the equation $x^2 + x + 1 = 3$.

### *B. Groth16*

Groth16 is a zk-SNARK protocol that efficiently proves knowledge of a solution to a mathematical problem without revealing the solution itself [2]. The protocol involves several key steps: first, the problem is encoded into an Arithmetic Circuit, where each gate corresponds to a constraint. Then, an execution trace is generated to track the computation process. The prover constructs polynomials using Lagrange interpolation, transforming the matrices representing the constraints into polynomial forms. These polynomials are combined using Quadratic Arithmetic Programs (QAPs), and a critical check is performed to ensure the correctness of the proof. Throughout this process, discrete logarithms are used to hide evaluation points, allowing the prover to carry out computations knowing these secret evaluation points in advance.

*Circuit:* : At the core of every Groth16 proof lies an Arithmetic Circuit, which serves as the basis for verifying a claim. This circuit is designed such that it is computationally difficult to reverse-engineer or solve given only the output, but easy to verify when both the input and output are known. To illustrate, consider the simple equation $x^2 + x + 1 = 3$ The prover will know the solution to this circuit (in this case, $x = 1$), and the verifier's task is to confirm the prover's knowledge of the solution without the prover revealing x, as depicted in Figure 2.

*Rank-1 Constraint System:*

1) Each gate in the circuit represents a constraint. In this case, the constraints are as follows: $t_1 = x * x$, $t_2 = t_1 + x$, and $t_3 = t_2 + 1$
2) In addition to the constraints, there is also an execution trace, which is a vector of the variables [1,x,$t_1$,$t_2$,$t_3$]. The execution trace represents the intermediate values calculated at each step of the circuit. For the given solution, the trace would be: [1,1,1,2,3].
3) Finally, the prover will have three matrices A, B, and C, the contents of which are shown in Figure 3.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Fig. 3. Matrices $A, B,$ and $C$ for the example Groth16 circuit.

These are not arbitrary matrices; they satisfy the equation $At * Bt = Ct$. By examining the first row of each matrix, we can verify that $A_0t * B_0t = C_0t$, which simplifies to $x * x = t_1$ representing our first constraint.

This system of equations is known as the Rank-1 Constraint System, which encodes the constraints in matrix form.

*Quadratic Arithmetic Programs:* Next, the matrices $A, B$, and $C$ are converted into polynomial vectors $A', B'$, and $C'$ through Lagrange Interpolation. These polynomials satisfy the relationships $A'(1) = A[0]$, $A'(2) = A[1]$, and so on. Given that $tA * tB = tC$, we can express this equation as polynomials using Quadratic Arithmetic Programs (QAPs) such that $P(x) = tA'(x) * tB(x) - tC(x) = 0$. This polynomial will have roots at $x = 1, 2, 3$, corresponding to the original solution values. Moreover, this equality implies the existence of a polynomial vector $H(x)$ such that $P(x)$ is divisible by $L(x)$, where $L(x) = (x-1)(x-2)(x-3)$ represents the product of the roots. Thus, the critical check in Groth16 is to verify that:

$$tA'(x) * tB'(x) = tC'(x) + H(x)L(x).$$

*Evaluation Through Schwartz-Zippel Lemma:*
*Lemma 3.1 (Schwartz-Zippel Lemma):* Let

$$f(x_1, x_2, \ldots, x_n) \in \mathbb{F}[x_1, x_2, \ldots, x_n]$$

be a non-zero polynomial over a field $\mathbb{F}$ with total degree $d \geq 0$. Let $S \subseteq \mathbb{F}$ be a finite subset [4].

Suppose $r_1, r_2, \ldots, r_n \in S$ are chosen independently and uniformly at random. Then,

$$\Pr[f(r_1, r_2, \ldots, r_n) = 0] \leq \frac{d}{|S|}.$$

Lemma 3.1 states that, given two polynomials evaluated at a random point $\tau$, if both polynomials yield the same result, there is a very high probability that they are equivalent. By leveraging discrete logarithms to obscure the evaluation points, the prover can perform their calculations for $A'(\tau), B'(\tau), C'(\tau)$, and $H(\tau)$ without ever uncovering the value of $\tau$.

## IV. Anonymous Authentication Protocol

### A. Design Principles

The goal of our design is to create the Client-Service authentication without revealing user information to the KDC. This inherently implies a protocol with a stronger privacy policy, as the KDC can no longer keep track of a log of all user-service usages.

However, the protocol must still allow services to correctly distinguish between different clients and prevent impersonations.

Moreover, we want to maintain all the desirable properties of Kerberos 5 as described in II; namely, passwords should never be transmitted in plaintext, and one can reuse an already-generated TGT numerous times, with the client only inputting the password once per TGT lifespan.

### B. Initial Design and Problems

The above principles might seem contradictory - how do we modify the Kerberos protocol such that services have more information about the client than the KDC itself? Our solution lies in zk-SNARKs.

Using a zk-SNARK, the client can send a proof that he knows a secret without revealing what the secret is. An initial idea is to leverage this property of verifying that a client is in the valid subset of clients, without revealing its username is as follows:

Attempt 1 - Roots of Polynomials

KDC Setup: The KDC chooses N large numbers and creates a polynomial with these N roots, ensuring that the polynomial is hard to factor. The KDC builds and publishes the circuit corresponding to this polynomial.

Client Setup: The KDC securely shares one of the roots $w$ of the polynomial with the client.

Authentication: The client sends a zk-SNARK proof that he knows $w$ such that $w$ is a root of the public polynomial. (Knows input $w$ of the circuit such that $P(w) == 0$).

This method allows clients that have securely set up with the KDC (hence, in the valid subset of clients) to authenticate without revealing information about themselves.

However, it has serious drawbacks. *First*, in order to remove, add, or modify clients, the polynomial needs to be modified, forcing the KDC to build a new circuit that it has to share with all the clients. Moreover, the KDC also needs to select new secret

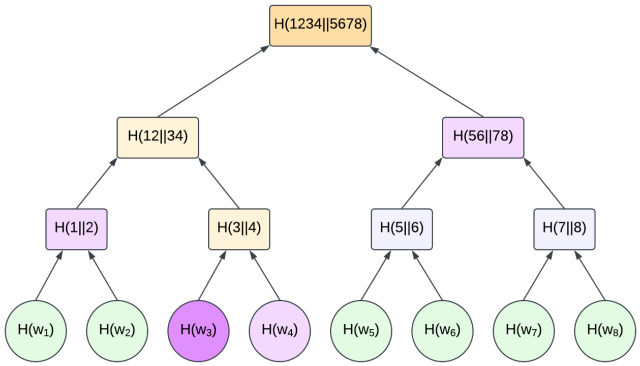


Fig. 4. Example of an N=8 Merkle Tree where the Merkle leaves are the commitments $H(w_i)$ of clients' secrets. The circuit *MerkleComputeRoot* takes as inputs (nodes colored purple) a leaf and the siblings of the path to compute the Merkle Root. In the example, *MerkleComputeRoot* takes as input $H(w_3)$ and the path of siblings $[H(w_4), H(1||2), H(56||78)]$ to output the root $H(1234||5678)$.

$w$s to all the clients: if it were to just modify a few roots at a time, it is computationally fast to find the GCD of the new polynomials with the previous polynomial and find what were the modified roots. Having to renew every secret for every modification to the client list makes this method completely impractical.

*Secondly*, while the KDC has authenticated the validity of possessing $w$, the symmetry of all clients implies that there is nothing that allow services to distinguish between clients. Most standard services require this distinction.

### C. Allowing efficient userbase scalability

Notice that the circuit of Attempt 1 has a structure that depends on the secrets. If a secret is modified, the circuit completely changes. Instead, we must choose a circuit such that the secrets are not built-in, but rather to be taken as possible inputs.

The output, however, must depend on all the possible secrets. If it depended on only a subset of secrets, a valid proof would leak knowledge on which clients have authenticated. Of course, any individual client only has access to their own secret; hence, the other inputs to the circuit must be values derived from the other client secrets. For convenience, this will be done with hashes.

To efficiently achieve these properties, consider a Merkle Tree where the leaves are commitments, $H(w_i)$, of the clients' secrets, $w_i$, as seen in Figure 4 [3]. The root of the Merkle Tree depends on hashes of all the secrets, and yet every single node of the tree can be made public without revealing the secrets. The following algorithm becomes apparent as a great candidate:

```
MerkleComputeRoot(leaf, siblings):
    node = leaf
    for sibling in siblings:
        node = H(node + sibling)
    return node
```

Where *siblings* is a list of the siblings of the nodes in the path from the leaf to the root. This leads us to a new circuit that solves our problem of adding clients. In particular, the client will now send a zk-SNARK proof that they possess a secret $w$ such that *MerkleComputeRoot*(H(w), siblings) outputs the root.

Attempt 2 - Merkle Tree

Client Setup: Client selects some secret $w$. In practice, $w$ can be $username||password$. Commit to this secret, $comm = H(w)$.

KDC Setup: The KDC creates a Merkle Tree with a fixed number N of leaves. Every leaf is a commitment $comm$ of some client secret $w$, or otherwise some default value (0). The Merkle Tree is made public.

Authentication: The client sends a zk-SNARK proof that he knows the inputs, namely $w$, to the circuit implementing *MerkleComputeRoot(H(w), siblings) == root*.

Notice that adding or removing clients requires modification of the leaves of the Merkle Tree, of which the KDC has full control of. Clients must continuously update siblings from the public Merkle Tree; however, this is a list of only $\log N$ nodes. Retrieving these values securely can be achieved with PIR, Oblivious RAM, or by serving the Merkle Tree content through some server who the client doesn't mind revealing their identity to.

### D. Services' distinguishability of clients

Most services require some way to distinguish between clients, something Attempt 2 does not support. The KDC receives proof that the client is valid. However, it cannot include the client's username in the ticket since, by design, the KDC does not know the username.

The solution is to have the KDC include a pseudonym that only the service can parse. We generate the pseudonym by concatinating a nonce and hashing the result. This gives us

$$nym = H(comm||nonce),$$

where $comm = H(w)$ is a leaf of the public Merkle Tree and a different nonce can be chosen randomly for every authentication request. The client sends $nym$ to the KDC with a proof that it was correctly computed with the same secret $w$ that generated the authentication proof. The KDC includes $nym$ in the ticket, but is unable to identify the client.

The client can reveal their identity to a service by sending both $comm$ and $nonce$ to the service, and the service can check that the $nym$ in the ticket corresponds to the correct hash.

Merkle Tree with Pseudonyms

Client Setup: (Unchanged from Attempt 2)

KDC Setup: (Unchanged from Attempt 2)

Authentication: The client chooses a random nonce and generates pseudonym

$$nym = H(w||nonce)$$

. Then, the client sends a zk-SNARK proof that they know the inputs, specifically $w$, to the circuit implementing

$$MerkleComputeRoot(H(w), siblings) == root$$

and

$$H(w||nonce) == nym.$$

Distinguishability: The KDC includes $nym$ in the ticket. To allow a service to be able to distinguish them from other clients, the client sends $comm$ and $nonce$, and the service checks that the concatenated value hashes to $nym$.

In addition, we want cooperating services to not be able to tell whether a given client has used both services (i.e. services $S_1$ and $S_2$ can't distinguish whether Alice is using $S_1$ and Bob is using $S_2$, or if Alice is using both $S_1$ and $S_2$). A solution to this problem is to utilize a different pseudonym for every service,

$$nym_S = H(H(w||service)||nonce).$$

However, note that this method requires sending many proofs to the KDC to generate the TGT. Nonetheless, this remains practical if you limit the number of pseudonyms in a TGT to only services that the client is most likely to visit in a given session. The entire Anonymous Authentication Kerberos Protocol is depicted in Figure 5.

## V. Implementation

We chose to build our modified version of Kerberos from scratch, focusing only on the essential components: a client and a Key Distribution Center (KDC).

*a) Client:* The client first attempts Zero Knowledge Authentication by creating a proof from the witness vector and two random field elements. Once authenticated, the client will connect to the server and complete key exchange with the KDC to generate a shared key. The KDC will present a ticket-granting ticket to the client, allowing them to connect to various services using Kerberos.

*b) KDC:* The KDC will authenticate the client by verifying the client's generated proof. If the KDC successfully authenticated the client, then it will complete the key exchange with the client. Once a shared key is established, then the KDC will present a ticket-granting ticket for the client to use.

### A. Zero Knowledge Authentication

We utilized the gnark library for our Groth 16 scheme. A circuit was decided by the client and server prior to authentication. The client is then given a solution to that circuit that the server has no knowledge of. Using the Groth 16 protocol of transforming the circuit into constraints and creating commitments based on the vectors of polynomial

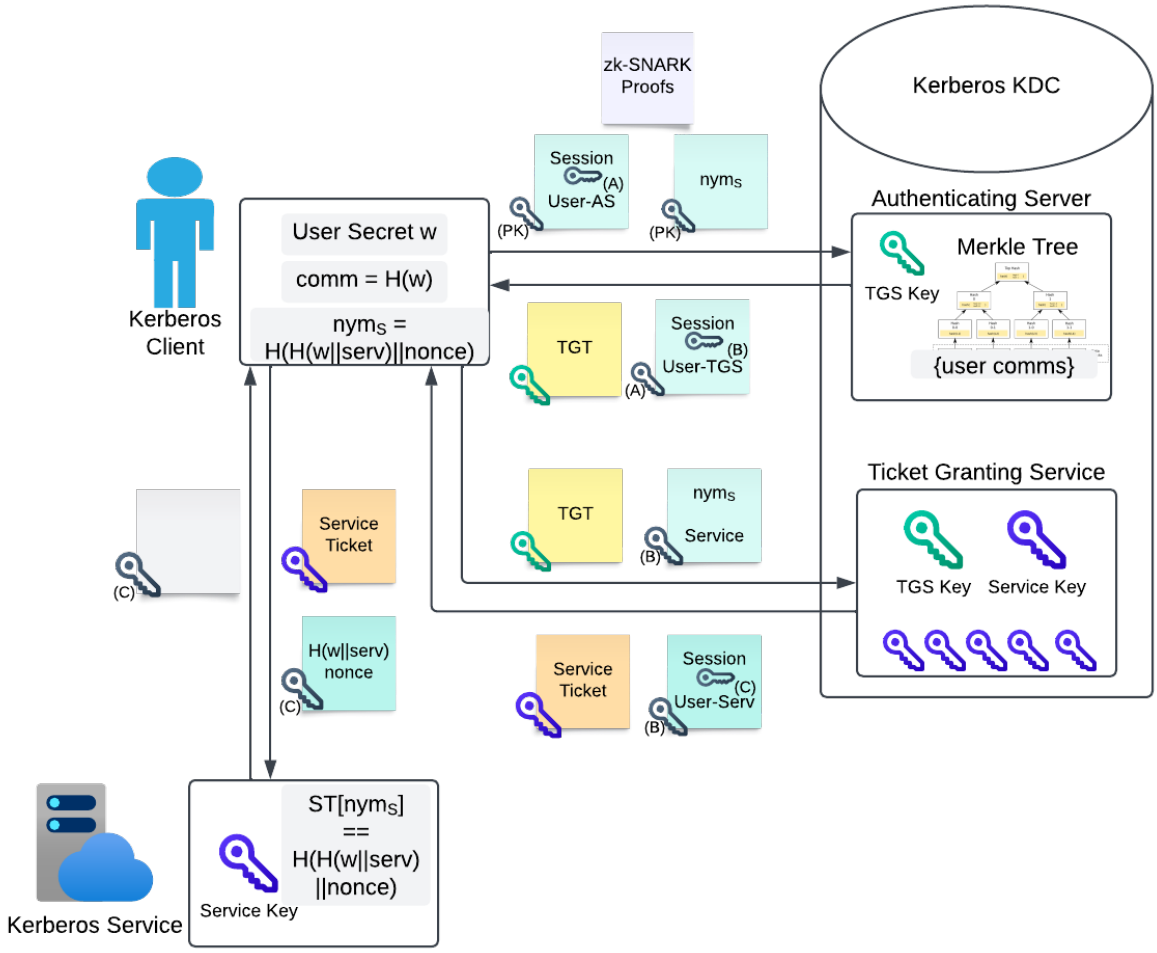


Fig. 5. Diagram of the Anonymous Kerberos Authentication Protocol. The client anonymously authenticates and provides a correct pseudonym by sending zk-SNARK proofs. The client also shares a session key for User-AS symmetric encryption. This communication is encrypted with a public key provided by the KDC during setup. If the proof is correctly verified, the AS replies with a ticket-granting-ticket (TGT) and an session key for User-TGS symmetric encryption. Possession of the TGT implies that the client was correctly validated by the AS; it also includes all relevant $nym_S$ and expiration values. Once a client wants to authenticate with a service, it requests a Service Ticket (ST) with the Ticket Granting Service (TGS) by sending the TGT along side the service it wants to connect. The client uses the ST to authenticate with the Service, and also sends H(w——service) and nonce for the Service to distinguish between users.

stated in section 3B, the client will then send his arguments over to the verifier. The verifier would then evaluate his commitments and will either approve or deny the client to continue with the Key exchange that initiates the next steps of communication.

### *B. Key Exchange*

We use a Diffie-Hellman Key Exchange protocol to generate a shared key between the client and KDC without needing to send the key across the network for confirmation, unlike how a classical password would work [5]. To implement this key exchange, we utilize the Go ”gob” package to create encoders and decoders to assist in processing the public keys for the client and KDC. The resulting shared key will play a similar role to the client’s password in the original Kerberos model.

### *C. Tickets*

The KDC generates a ticket granting ticket which is encrypted using AES golang library and the private key of the TGS, hence only the TGS will be capable of decrypting. When requesting access

to a service, the TGS encrypts the ticket with AES and the key of the service.

## Conclusion

The Kerberos authentication scheme has had far-reaching impact since its creation. Originally developed by MIT to serve its student body, Kerberos has since been adopted more broadly and is now supported by major operating systems, including Windows and macOS. Its design is intuitive: a centralized KDC handles client authentication and facilitates secure, direct access between clients and services.

Inspired by the mathematical elegance of zero-knowledge authentication schemes, particularly Groth16, our goal is to develop a more secure and privacy-preserving variant of the Kerberos protocol. Unlike the traditional model, our revised scheme ensures that the central KDC has no ability to link users to their service activity. Moreover, even if the KDC were to be compromised, user credentials would remain secure and protected. These enhancements provide clients with stronger privacy guarantees and greater confidence in the security of their personal information.

## Future Work

Currently, our proposed scheme incurs greater computational overhead compared to the traditional Kerberos protocol. One major source of this increased cost is the zero-knowledge authentication protocol, which introduces complexity that grows linearly with the size of the underlying arithmetic circuit. A more significant bottleneck arises in the management of user information, which is stored in a Merkle Tree. Any update, addition, or deletion of user data fundamentally alters the content of the tree, resulting in changes to the authentication paths and increased computational burden. As part of future work, we aim to further refine our design to reduce these computational costs and bring performance closer to that of the original Kerberos system, while retaining the added privacy and security benefits.

## Contributions and Acknowledgments

All three group members contributed equally to every aspect of the project, from implementing the protocol modifications in Go to writing and editing this paper.

We would like to thank the instructors and course staff for their guidance throughout this project and their excellent instruction, which gave us the foundational knowledge to explore and apply these advanced concepts to Kerberos.

## References

[1] J. Kohl and C. Neuman, " The Kerberos Network Authentication Service (V5)," Network Working Group, RFC1510, September 1993.

[2] J. Groth, "On the Size of Pairing-based Non-interactive Arguments," in International Association for Cryptologic Research, Springer-Verlag, 2016.

[3] O. Kuznetsov, A. Rusnak, A. Yezhov, et al. "Merkle Trees in Blockchain: A Study of Collision Probability and Security Implications," in Internet of Things Volume 26, July 2024.

[4] A. Atserias and I. Tzameret. "Feasibly Constructive Proof of Schwartz-Zippel Lemma and the Complexity of Finding Hitting Sets," in arXiv, November 2024.

[5] N. Li. "Research on Diffie-Hellman key exchange protocol," in 2nd International Conference on Computer Engineering and Technology, June 2010.

[6] C. Chen, E. Hong, and L. Turino, "ZK-Kerberos".