TSNC: A Timestamp-Driven Stochastic Neural Architecture for Deterministic yet Irreversible Password Hashing

Yongqian Xiao[1], Mingfeng Su[1,*], Hao Chen[1], Jin Li[1,*], Si Zhou[2]
[1]School of Computer Science, Hunan First Normal University, Changsha, 410205, P. R. China
[2]Hunan Engineering Technician College, Sinohydro Engineering Bureau 8 Co., Ltd., Changsha, 410000, P. R. China
E-mail: xiaoyongqian@hnfnu.edu.cn, mfsu@hnfnu.edu.cn, goldliclass@163.com

Keywords: Stochastic neural networks, password hashing, cryptographic hash function, temporal-stochastic, side-channel resistance

Received: January 8, 2025

Traditional unsalted hash functions and static neural hashing schemes produce identical outputs for identical inputs, making them vulnerable to rainbow-table and precomputation attacks when no per-account diversification mechanism is used. To address this, this paper presents the Temporal-Stochastic Neural Cipher (TSNC), a novel framework that achieves both input-dependent stochastic transformation and reproducible deterministic verification. TSNC removes the training phase by deriving SHA-256/SHA-512-based seeds from the account creation timestamp and password to initialize a training-free stochastic neural pipeline. The framework establishes a hashing pipeline through three logically cascaded modules: (1) timestamp-driven dynamic dictionary construction that ensures temporal uniqueness for character embeddings; (2) sequential feature extraction utilizing recurrent transformations with layer normalization to capture inter-character dependencies; and (3) feed-forward stochastic neural blocks with input-dependent depth and width for nonlinear irreversible transformation. Theoretical analysis confirms the method's computational irreversibility and intractability against reverse engineering. Comprehensive simulations validate TSNC's security properties, demonstrating zero observed collisions in 10^7 empirical trials, a strong avalanche effect with output feature vectors exhibiting near-zero cosine similarity under minimal input perturbations, and temporal orthogonality—properties that reduce the practicality of rainbow-table and precomputation attacks. Furthermore, TSNC achieves timing-stabilized execution of approximately 200 ms across password lengths from 1 to 1000 characters, with an intermediate memory footprint (∼7.84 MB). To promote reproducibility and further research, the source code of the proposed algorithm is publicly available at https://github.com/yongqianxiao/TSNC.

Povzetek: Študija predstavi TSNC, ki s časovnim žigom in stohastičnimi nevronskimi bloki oblikuje deterministično preverljivo, a ireverzibilno funkcijo za zgoščevanje gesel. Okvir zmanjšuje tveganje mavričnih tabel in predizračunanih napadov ter izboljšuje časovno stabilnost preverjanja.

1 Introduction

In an era characterized by pervasive digitalization, passwords remain a primary authentication mechanism for protecting user data, system integrity, and access control [7, 28, 8]. However, they are increasingly challenged by brute-force guessing, rainbow-table attacks, precomputation, collision attacks, and AI-driven assaults [21, 12, 2, 9]. Traditional cryptographic hash functions such as MD5 and SHA are efficient but deterministic, enabling large-scale lookup and precomputation attacks when no diversification mechanism is used [21, 12], which has contributed to serious password-database breaches [2]. Moreover, quantum search may reduce the effective margin of brute-force resistance, motivating password hashing schemes with higher per-guess cost and stronger precomputation resistance [9].

To mitigate these vulnerabilities, memory-hard functions such as bcrypt [25], PBKDF2 [13, 19], scrypt [24], and Argon2 [6] increase the computational and memory costs of offline guessing [4]. Nevertheless, their resource intensity may become a bottleneck in high-throughput deployments [4], and careful implementation is still needed to reduce timing and power side-channel leakage [16, 15]. In distributed, IoT, and cloud environments, static hashing mechanisms may also face replay or synchronization-related risks when dynamic time-variant factors are absent [5, 17, 20]. Thus, purely deterministic mechanisms remain less suitable for time-correlated authentication scenarios [5].

In response to these challenges, the intersection of deep learning and cryptography has emerged as a fertile research frontier [29, 18, 1]. Neural networks (NNs) offer unique advantages, including high-dimensional non-linearity and adaptive feature extraction. Randomly initialized neural networks can act as nonlinear mixing operators, producing statistically hard-to-invert outputs. Training-free stochas-

tic networks further avoid the gradient-inversion risks of trained models while retaining nonlinear diffusion capability.

Extensive studies have also applied neural models to security-related sequential data processing, including LSTM-based key generation in vehicular networks [30], error-resilient encrypted traffic detection [31], and interpretable flow classification [26]. Neural cryptography has shown that neural networks can learn communication-protection strategies without prespecified cryptographic algorithms. In password security, GAN-based models such as PassGAN learn password distributions for stronger guessing attacks, while neural methods have also been used for password-strength evaluation [10, 3, 14]. These studies motivate hashing mechanisms resistant to AI-assisted password analysis.

CNN-based architectures have been explored in security tasks and deep hashing. However, conventional deep hashing is often similarity-preserving, whereas password hashing requires minimally different inputs to produce statistically unrelated outputs.

Despite these advancements, neural networks for password hashing remain limited by fixed trained parameters. Existing schemes such as deep self-learning hashing [27] and RNN-based text hashing [32] usually require training and become static mappings after deployment: identical passwords yield identical feature vectors, and the feature dimensionality remains fixed. This reintroduces dictionary and rainbow-table risks, while training cost and possible gradient inversion further hinder lightweight or real-time authorization deployment [14]. Consequently, a training-free, dynamic, and unpredictable neural hashing framework is needed.

Formally, TSNC aims to satisfy four requirements: (i) *deterministic verifiability*, where the same password and stored timestamp reproduce the same authentication vector; (ii) *precomputation resistance*, where identical passwords with different timestamps yield decorrelated outputs; (iii) *avalanche behavior*, where small input changes cause global output changes; and (iv) *timing stability*, where latency is independent of password length and internal state [16].

The considered threat model includes offline dictionary/rainbow-table attacks, timing side-channel attacks, and gradient-based inversion attacks. TSNC is a standalone password hashing scheme rather than an augmentation layer and requires no pre-trained model. We evaluate three hypotheses: (H1) no observed collisions in 10^7 empirical trials; (H2) stable hashing latency across password lengths under the tested configuration; and (H3) decorrelated outputs for identical passwords registered at different timestamps.

To address these limitations, this paper proposes an innovative framework: the Temporal-Stochastic Neural Cipher (TSNC). Unlike traditional static architectures, TSNC leverages account creation timestamps as dynamic seeds to generate cryptographic keys for stochastic neural networks. This approach reconciles input-dependent stochastic transformation with reproducible deterministic verification by ensuring that the hashing topology itself adapts to the user's temporal context. TSNC exhibits several distinctive features: (1) **Dynamic Orthogonality**: Hashed feature vectors of identical passwords across different accounts are statistically independent due to unique timestamps, effectively neutralizing rainbow table attacks; (2) **Training-Free Efficiency**: It eliminates the need for backpropagation, relying solely on forward inference through randomly initialized blocks, which sustains high computational efficiency; (3) **Irreversibility**: The combination of dynamic dictionaries and stochastic weights increases the difficulty of reverse engineering under the considered threat model.

Table 1 summarizes the key properties of representative password hashing schemes and TSNC, highlighting the distinctions in terms of determinism, dynamic salting, training requirement, side-channel resistance, and architectural flexibility.

The main contributions of this paper are summarized as follows:

1. **A timestamp-driven dynamic hashing architecture**: We introduce a mechanism where account creation timestamps and passwords jointly generate cryptographically secure random seeds. This design avoids generating an additional random salt, since the stored timestamp serves as a per-account diversification factor, and ensures that identical passwords produce statistically independent authentication vectors across different registration times.

2. **Multi-stage stochastic transformation for irreversibility**: We design a three-stage pipeline comprising character embedding, sequential extraction, and stochastic neural blocks. By cascading randomness with non-linear operations, the framework ensures that password reverse-engineering is computationally intractable, providing resistance against brute-force attempts.

3. **Adaptive adjustment with side-channel defense**: The framework dynamically adjusts structural parameters—such as neural block count and feature dimensions—to prevent fixed-pattern vulnerabilities. Furthermore, it incorporates timing-stabilized execution mechanisms to mitigate timing and gradient inversion attacks, making it suitable for real-world authentication systems.

2 Methodology

In this section, we propose the Temporal-Stochastic Neural Cipher (TSNC), a novel password hashing framework that uniquely leverages stochastic neural networks to reconcile stochastic transformation with deterministic verification. The approach combines account creation timestamps

Table 1: Comparison of representative password hashing schemes and TSNC.

Scheme	Deterministic	Dynamic Salt /Timestamp	Training-Free	Side-Channel Resistant	Dynamic Topology	Neural Based
MD5 / SHA-256	✓	×	✓	×	×	×
bcrypt [25]	✓	Static salt	✓	Partial	×	×
PBKDF2 [13, 19]	✓	Static salt	✓	Partial	×	×
scrypt [24]	✓	Static salt	✓	Partial	×	×
Argon2 [6]	✓	Static salt	✓	Partial	×	×
Deep Hashing [27]	✓	×	×	×	×	✓
RNN Hashing [32]	✓	×	×	×	×	✓
TSNC (Ours)	✓	**Timestamp**	✓	✓	✓	✓

with password content to generate deterministic yet unpredictable feature vectors. This framework ensures that even identical passwords yield distinct hashed representations across different user accounts, enhancing security against rainbow table attacks and credential stuffing. Furthermore, it relies solely on forward inference through randomly initialized neural networks, eliminating the need for computationally expensive training procedures while maintaining its security properties.

2.1 System overview

The general architecture of the proposed TSNC algorithm is depicted in Fig. 1 (a). The framework accepts a password string p_w and a timestamp T_s as inputs, outputting an authentication vector that is stored locally for authorization validation. For login verification, the account creation timestamp T_s is stored with the authentication vector, analogous to a salt. The system retrieves T_s, recomputes the vector from the submitted password and stored T_s, and performs exact comparison; no timestamp drift tolerance or fuzzy matching is required. A critical design choice in our system is the source of randomness; specifically, the timestamp, combined with the password information, serves as the foundation for generating cryptographically secure random seeds through SHA-512 hashing, providing stronger security guarantees than conventional pseudo-random number generators [23, 22].

The hashing pipeline is structured into three logically cascaded components: (1) Cryptographically Secure Initialization: Generating a sequence of secure seeds from the timestamp-password pair to initialize a dynamic dictionary. (2) Sequential Context Extraction: Processing the password feature matrix through recurrent transformations to capture inter-character dependencies. (3) Stochastic Neural Network Processing: Transforming the feature vector through a series of neural blocks with dynamic topology to ensure irreversibility.

2.2 Cryptographically secure weight generation and character embedding

The first component of TSNC involves the joint utilization of the account creation timestamp T_s and password p_w to construct a Cryptographically Secure Pseudorandom Number Generator (CSPRNG). This generator produces a sequence of seeds $s_i, i \in [0, N_s]$ used for deriving weight matrices in subsequent steps. This dual-dependency ensures that the randomness is both temporally unique and password-specific, significantly enhancing the security of the hashing process.

Given a timestamp seed T_s and password p_w, the combined seed for the CSPRNG is computed as:

$$S_c = (T_s \oplus H(p_w)) \bmod (2^{32} - 1), \tag{1}$$

where $H(\cdot)$ denotes the SHA-256 cryptographic hash function, and $\oplus$ represents the XOR operation. Based on S_c, a sequence of cryptographically secure seeds is generated through SHA-512 to construct the dictionary and stochastic neural network blocks.

Subsequently, we construct a dynamic character embedding dictionary $\mathcal{D}_{T_s}$ given a timestamp seed T_s and feature dimension d:

$$\mathcal{D}_{T_s} = \left\{ c_i \mapsto v_i \mid c_i \in \mathbb{C}, v_i \in \mathbb{R}^d \right\}, \tag{2}$$

where $\mathbb{C}$ is the set of all possible characters (including alphanumeric, special symbols, and Unicode characters), and v_i represents the embedding vector for character c_i. Each v_i is deterministically derived from the CSPRNG-driven weights seeded by S_c. Consequently, identical characters produce identical embeddings if and only if they share the same account creation timestamp and password.

For a password string with length ℓ, each character $p_{w,i} \in \mathbb{C}$ is transformed into a feature vector via the constructed dictionary:

$$\phi(p_{w,i}) = \sigma(\mathcal{D}_{T_s}(p_{w,i})), \tag{3}$$

where $\phi(p_{w,i}) \in \mathbb{R}^d$ for $i \in [1, \ell]$ denotes the embedding feature vector. It is essential to incorporate all possible password characters into the dictionary construction to improve the embedding security, as characters outside $\mathbb{C}$ would otherwise yield identical feature vectors.

2.3 Sequential feature transformation

Following character embedding, we perform a sequential transformation to capture the dependencies between characters. This step ensures that the permutation of characters fundamentally alters the final hashed representation.

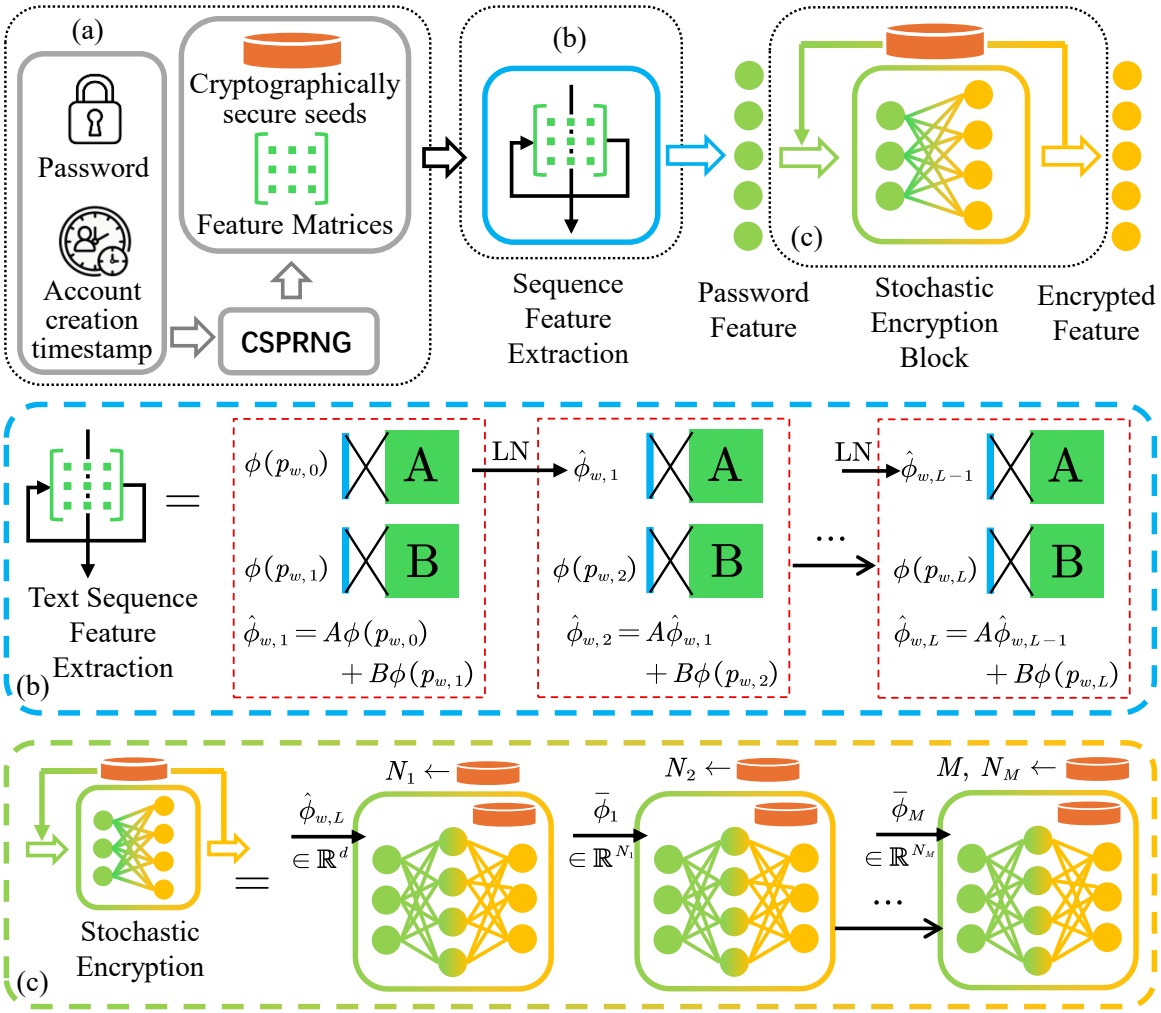

Figure 1: Schematic overview of the TSNC hashing system. (a) **Seed Generation & Embedding**: A CSPRNG driven by the timestamp-password pair initializes a dynamic dictionary. (b) **Sequential Extraction**: Recurrent transformations with LayerNorm capture inter-character dependencies. (c) **Stochastic Transformation**: A chain of randomly initialized neural blocks performs the final non-linear mapping to ensure high entropy and irreversibility.

Given the embedded feature vectors $\phi(p_{w,i})$ for $i = 1, \cdots, L$, we initialize two random matrices $A \in \mathbb{R}^{d\times d}$ and $B \in \mathbb{R}^{d\times d}$ using distinct seeds generated from the CSPRNG:

$$A \sim \mathcal{U}(0,1) \cdot \frac{\alpha}{\sqrt{d}}, \; B \sim \mathcal{U}(0,1) \cdot \frac{\alpha}{\sqrt{d}} \tag{4}$$

where $\mathcal{U}(0,1)$ denotes a uniform distribution, and α is a scaling factor. The normalization term $\frac{\alpha}{\sqrt{d}}$ draws inspiration from Xavier/Glorot [11] initialization principles. It maintains signal magnitudes within the responsive regions of nonlinear activation functions, preventing saturation effects that would otherwise diminish feature distinctiveness during recursive transformations. Simultaneously, it regulates the evolution rate of the feature representation in the sequential processing pipeline, establishing an equilibrium between computational stability and cryptographic variability.

To prevent hashing latency from scaling linearly with raw password length, we preprocess the password using a cryptographic hash to output a fixed-length string of size L prior to feature extraction. As shown in Fig. 1 (b), the sequence is processed using a recurrent formula with layer normalization:

$$\hat{\phi}_{w,i} = \text{LayerNorm}(A\hat{\phi}_{w,i-1} + B\phi(p_{w,i})), \tag{5}$$

where $\hat{\phi}_{w,i}$ indicates the i-th feature vector after i recursive transformations; $\hat{\phi}_{w,0} \in \mathbb{R}^d$ is generated by the CSPRNG with the first seed; $i = 1, 2, \cdots, L$, and LayerNorm is defined as:

$$\text{LayerNorm}(x) = \alpha_1 \cdot \frac{x-\mu}{\sqrt{\sigma^2+\epsilon}} + \beta_1, \tag{6}$$

with μ and σ^2 being the mean and variance of x, $\epsilon = 10^{-5}$ for numerical stability, and α_1, β_1 as the scaling and shifting parameters. The output of this process, $\hat{\phi}_{w,L} \in \mathbb{R}^d$, is a

feature vector that captures the sequential information of the password.

LayerNorm plays a critical role by preventing numerical instability; without it, repeated matrix multiplications could cause feature values to grow exponentially, potentially undermining security. By stabilizing feature distributions, it ensures consistent cryptographic properties regardless of password length while introducing additional nonlinearity.

2.4 Stochastic neural network processing

To further enhance security and irreversibility, the feature vector is processed through a series of stochastic neural network blocks. Each block $\mathcal{B}$ is a feed-forward neural network whose weights, biases, and output dimensions are randomly initialized based on the secure seeds.

The transformation is defined as

$$\bar{\phi}_i = \mathcal{B}_i(\bar{\phi}_{i-1}), \tag{7}$$

where $\bar{\phi}_i \in \mathbb{R}^{N_i}$ and $\bar{\phi}_{i-1} \in \mathbb{R}^{N_{i-1}}$ indicates the output and input feature vector of the i-th stochastic block, respectively; and $\bar{\phi}_0 = \hat{\phi}_{w,L} \in \mathbb{R}^d$. Specifically, each block consists of two hidden layers with tanh activation functions:

$$\begin{aligned} h_{1,i} &= \tanh(W_{1,i}^\top \cdot \bar{\phi}_{i-1} + b_{1,i}) \\ h_{2,i} &= \tanh(W_{2,i}^\top \cdot h_{1,i} + b_{2,i}) \\ \bar{\phi}_i &= W_{f,i}^\top \cdot h_{2,i} + b_{f,i} \end{aligned} \tag{8}$$

where $i \in [1, M]$ denote the number of stochastic blocks; $W_{1,i} \in \mathbb{R}^{N_{i-1} \times K}$, $W_{2,i} \in \mathbb{R}^{K \times K}$, $W_{f,i} \in \mathbb{R}^{K \times N_i}$ and $b_{1,i} \in \mathbb{R}$, $b_{2,i} \in \mathbb{R}$, $b_{f,i} \in \mathbb{R}$ are randomly generated weights and biases by the cryptographically secure seeds s_i. $K = 2N_{i-1}$ is the number of neurons in hidden layers. $h_{1,i}$ and $h_{2,i}$ are the outputs of hidden layers of the i-th stochastic block.

A key innovation of TSNC is its dynamic topology. To enhance the irreversibility, the number of stochastic blocks M is randomly generated based on the intermediate feature $\hat{\phi}_{w,L}$:

$$M = \lfloor \sigma(w_m^\top \cdot \hat{\phi}_{w,L}) \cdot (M_{\max} - M_{\min}) + M_{\min} \rfloor, \tag{9}$$

where $w_m \in \mathbb{R}^d$ is a random vector with the seed $s_m = s_i + 1$; $\lfloor \cdot \rfloor$ and σ denote the floor function and sigmoid function, respectively.

Similarly, the number of neurons in the output layers of each block, N_i, is determined based on the input feature vector $\bar{\phi}_{i-1}$ via Eq. (10), ensuring that the dimensionality of the feature space evolves dynamically:

$$N_i = \lfloor \sigma(w_{N,i}^\top \cdot \bar{\phi}_{i-1}) \cdot (N_{\max} - N_{\min}) + N_{\min} \rfloor, \tag{10}$$

where $\bar{\phi}_0 = \hat{\phi}_{w,L}$; $N_{\max}$ and $N_{\min}$ are the maximum and minimum allowed number of neurons in output layers; $w_{N,i} \in \mathbb{R}^{N_{i-1}}$ is generated with the seeds with $W_{1,i}$, $W_{2,i}$, and $W_{f,i}$. For the last block, we enforce $N_M = d$. Furthermore, the seed for the subsequent block is derived by combining s_{i+1} with the current block's output via a hash function H_i, creating a dependency chain that resists reverse engineering:

$$s_{i+1} = H_i(s_{i+1}, \bar{\phi}_i), \tag{11}$$

where H_i is a cryptographic hash function. The final irreversible feature vector is denoted as $\psi_{p_w} = \bar{\phi}_M$.

Remark 1 *Convolutional layers are not adopted because TSNC uses abstract feature vectors without spatial locality, and its input-dependent block count M and output dimension N_i are more naturally implemented by fully connected stochastic transformations.*

2.5 Side-channel defense

To mitigate timing side-channel attacks, our algorithm enforces constant-time execution through a two-fold mechanism. First, input passwords are normalized via SHA-256 hashing to eliminate length-dependent processing variations. Second, to counterbalance the timing discrepancies introduced by the dynamic depth and dimensionality of the stochastic blocks, we implement a computational padding strategy. TSNC quantifies the gap between the current instance's computational load and the theoretical upper bound (determined by the maximum block count and feature dimensions). It then executes supplementary dummy matrix operations to bridge this gap, ensuring that the total hashing latency remains statistically independent of the specific password structure and internal state.

The implementation pipeline is summarized in Algorithm 1. The proposed algorithm ensures determinism, unpredictability, sensitivity, and one-way transformation.

3 Analysis of irreversibility

In this section, we provide a theoretical analysis of the computational irreversibility of the proposed TSNC framework. Unlike traditional static cryptographic primitives, TSNC constructs a dynamic, data-dependent transformation pipeline. The irreversibility is established through a hierarchical argumentation, progressing from the structural non-invertibility of the dynamic topology to the computational intractability of the parameter space, and finally, the robustness against specific cryptanalytic vectors.

3.1 Structural irreversibility

Proposition 1 *The mapping function $f : (p_w, T_s) \mapsto \psi_{p_w}$ constructed by TSNC constitutes a dynamic one-way transformation where the transformation structure is dependent on the input p_w itself, rendering the inversion process logically circular and structurally ambiguous.*

Algorithm 1 TSNC password hashing

Require: Password string p_w, account creation timestamp T_s, embedding dimension d, scaling factor α, $M_{\max}$, $M_{\min}$, $N_{\max}$, $N_{\min}$, N_s;
Ensure: hashed feature vector ψ_{p_w};
/* Secure Seed Generation */
Calculate combined seed S_c by (1);
Construct a CSPRNG to generate a sequence of secure seeds $\{s_0, s_1, \ldots, N_s\}$ based on S_c;
Initialize $c = 0$;
/* Character Embedding */
Construct dictionary $\mathcal{D}_{T_s}$ using seed s_{N_s-c}, $c = c+1$;
Preprocess p_w to fixed length L via cryptographic hash;
Embed characters to obtain $\phi(p_{w,i})$ by (3) for $i \in [1, L]$;
/* Sequential Transformation */
Initialize matrices A and B according to (4) using seeds $s_{N_s-c} - 1$ and $s_{N_s-c} - 2$;
Initialize $\hat{\phi}_{w,0}$ using seed $s_{N_s-c} - 3$, and $c = c + 1$;
for $i = 1$ to L **do**
  $\hat{\phi}_{w,i} \leftarrow \text{LayerNorm}(A\hat{\phi}_{w,i-1} + B\phi(p_{w,i}))$;
end for
/* Stochastic Neural Network */
$\bar{\phi}_0 \leftarrow \hat{\phi}_{w,L}$;
Determine the number of stochastic blocks M by (9) using seed s_{N_s-c}, and $c = c + 1$;
for $i = 1$ to M **do**
  $N_{i-1} \leftarrow$ dimension of $\bar{\phi}_{i-1}$;
  Generate weights using s_{N_s-c}: $W_{1,i} \in \mathbb{R}^{N_{i-1}\times K}$, $W_{2,i} \in \mathbb{R}^{K\times K}$, $W_{f,i} \in \mathbb{R}^{K\times N_i}$, $w_{N,i} \in \mathbb{R}^{N_{i-1}}$;
  Generate biases: $b_{1,i} \in \mathbb{R}^K$, $b_{2,i} \in \mathbb{R}^K$, $b_{f,i} \in \mathbb{R}^{N_i}$;
  if $i < M$ **then**
    Determine output dimension N_i by (10);
  else
    $N_i \leftarrow d$;
  end if
  $K \leftarrow 2 \cdot N_{i-1}$; {Hidden layer size}
  Calculate $\bar{\phi}_i = \mathcal{B}_i(\bar{\phi}_{i-1})$ by (8);
  $c = c + 1$;
  Update seed $s_{N_s-c} \leftarrow H_i(s_{N_s-c}, \bar{\phi}_i)$ by (11);
end for
$\psi_{p_w} = \bar{\phi}_M$;

Analysis 1 *The structural irreversibility stems from three intertwined architectural mechanisms:*

1. ***Input-Dependent Dynamic Topology:*** *A critical innovation of TSNC is that the neural network architecture itself—specifically the number of blocks M (Eq.* (9)*) and the dimensionality of each layer N_i (Eq.* (10)*)—is dynamically determined by the intermediate features of the password. Consequently, an attacker attempting to reverse-engineer the input ψ_{p_w} faces a causality dilemma: to invert the neural network, one must know its precise topology (weights, biases, depth, and width); however, this topology is stochastically generated based on seeds derived from the password p_w itself. This circular dependency creates a logical deadlock for any inversion algorithm that does not possess the original password.*

2. ***Non-Linear Information Compression:*** *The hashing process employs a cascade of non-linear operations, including $tanh$ activations, LayerNorm (Eq.* (6)*), and dimension reduction operations. These transformations effectively compress the input space into the feature space, creating a many-to-one mapping. This entropy maximization ensures that for a given output ψ_{p_w}, the set of potential pre-images is theoretically large and computationally indistinguishable, thereby preventing unique reconstruction of the plaintext.*

3. ***Cascading Avalanche of Randomness:*** *The seed update mechanism (Eq.* (11)*) establishes a strictly sequential dependency chain. The randomness of the $(i+1)$-th block depends not only on the initial timestamp but also on the non-linear output of the i-th block. This design ensures that the cryptographic entropy diffuses rapidly throughout the network layers, making the relationship between the input characters and the final feature vector highly complex and statistically uncorrelated.*

3.2 Computational intractability

Proposition 2 *Inverting the hashing function is computationally intractable due to the combinatorial explosion of the parameter space and the cryptographic strength of the seed generation process, even under the assumption that the account creation timestamp T_s is public.*

Analysis 2 *The computational hardness is guaranteed by the following factors:*

1. ***Cryptographically Secure Parameter Generation:*** *The neural network weights are not learned parameters but are generated via a CSPRNG seeded by S_c. Since S_c is derived from the SHA-256 hash of the password and timestamp (Eq.* (1)*), recovering the weights requires solving the pre-image problem of the SHA-256 function. Even if T_s is known, the unknown p_w renders the seed S_c unpredictable.*

2. ***Combinatorial Explosion of the Search Space:*** *The theoretical search space is the Cartesian product of the password space and the timestamp space. Let $|\mathcal{C}|$ be the character set size, ℓ the password length, and $|\mathcal{T}|$ the effective timestamp space (e.g., millisecond precision over decades). The total state space is defined as $\mathcal{S}_{total} \approx |\mathcal{T}| \times \sum_{i=1}^{\ell_{\max}} |\mathcal{C}|^i$. While a specific T_s might be known in a database breach, this multiplicative factor $|\mathcal{T}|$ is critical for resisting pre-computation attacks. Unlike fixed-structure networks, TSNC requires the attacker to simulate the dynamic network generation process for every candidate pair (p_w, T_s). With the hidden layer size K and output dimension N_i varying dynamically, the computational cost of verifying a single guess is significantly higher than that of standard hash functions, imposing a prohibitive computational burden on attackers. Thus, under the considered*

offline setting, an attacker must test candidate passwords by regenerating the SHA-256/SHA-512-derived seeds and the corresponding dynamic neural pipeline. The unpredictability of these seeds relies on the pre-image resistance of SHA-256, for which no practical generic inversion attack is known [9].

3.3 Robustness against cryptanalytic attacks

Proposition 3 *The TSNC framework provides resistance under the considered threat model against both classical cryptanalysis (Rainbow Tables, Differential Attacks) and modern AI-driven attacks (Gradient Inversion).*

Analysis 3 *We analyze the defense mechanisms against three primary attack vectors:*

1. ***Defense Against Rainbow Table and Precomputation:*** *The account creation timestamp T_s functions as a mandatory, high-entropy dynamic salt. As demonstrated in the simulation section, identical passwords generate orthogonal feature vectors under different timestamps. This "Temporal Orthogonality" invalidates precomputed lookup tables (Rainbow Tables), as an attacker would need to build a separate table for every distinct timestamp (millisecond precision), which is storage-infeasible.*

2. ***Immunity to Gradient-Based Inversion:*** *Traditional neural network attacks often utilize gradient descent to optimize an input that matches the target output. However, TSNC is immune to such attacks for two reasons: (a) Discontinuous Loss Landscape: The discrete nature of the dynamic topology (integer changes in M and N_i based on thresholding) makes the function non-differentiable with respect to the input; (b) Stochastic Weights: The weights are not fixed values optimized for a task but are pseudo-random variables dependent on the input. Therefore, gradients cannot be backpropagated to update or approximate the input password.*

3. ***Resistance to Differential Cryptanalysis:*** *The sequential feature transformation (Eq. (5)) and the multi-stage neural blocks support a strong avalanche-like effect. A single bit change in the input password alters the initial seed S_c, which completely re-initializes the dictionary $\mathcal{D}_{T_s}$ and all subsequent network weights. This global decorrelation prevents attackers from exploiting similarities in output vectors to deduce relationships between input passwords.*

3.4 Comparison with salted and probabilistic hashing schemes

Salted password hashing schemes such as bcrypt [25] and Argon2 [6] use a stored salt to diversify identical passwords across accounts. TSNC follows the same verification principle by storing the timestamp as a per-account diversification value, but differs in its internal computation: the timestamp-password pair determines both the random seeds and the stochastic neural topology. Thus, even when T_s is known after a database breach, each password guess requires regenerating the corresponding dynamic pipeline.

Compared with existing neural hashing schemes [27, 32], which rely on fixed trained weights, TSNC is training-free and input-dependent. Its main distinction is the combination of per-account structural diversification, absence of learned model parameters, and timing-stabilized execution, which helps mitigate timing leakage in practice [16].

4 Simulation

In this section, we conduct a rigorous empirical evaluation of the proposed temporal-stochastic neural cipher (TSNC). The objective is to validate its cryptographic robustness from four progressive dimensions: statistical indistinguishability, temporal-stochastic orthogonality, sensitivity to input perturbations (avalanche effect), and resistance to physical implementation attacks (side-channel analysis).

4.1 Experimental setup

All simulations were executed on a workstation equipped with an Intel Core i9-11900KF CPU (3.5GHz) and 64GB RAM, running the Windows 11 operating system. The structural parameters of the TSNC framework employed in our experiments are detailed in Table 2.

Table 2: Parameter settings of the proposed TSNC framework.

Param	T_s	$N_{\max}$	$N_{\min}$	$M_{\max}$	$M_{\min}$
Value	1761106370865	400	32	10	3
Param	α	α_1	β_1	d	N_s
Value	0.1	1	0	128	100

To ensure a comprehensive evaluation across varying entropy levels, we constructed six distinct password datasets:

- **Simple**: Common passwords such as "password", "123456";
- **Complex**: Strong passwords with mixed characters, e.g., "P@ssw0rd!", "C0mpl3x#Pwd";
- **Unicode**: Passwords containing non-ASCII characters, e.g., "café♥", "über123";
- **Similar pairs**: Password pairs with minor variations (swaps, additions, substitutions), e.g., ("secure" vs "secrue") and ("admin123", "admin1234").
- **Different lengths**: Randomly generated passwords with lengths from 1 to 1000 characters;
- **Special chars**: Passwords composed entirely of special characters, e.g., "!@#$%^&*()", " !@#$%^&*()_+".

Trained deep hashing baselines [27, 32] are not empirically compared because they require supervised training and fixed learned weights, whereas TSNC is training-free and dynamically generated.

4.2 Statistical uniformity and feature whitening

A desirable property of password hashing outputs is the absence of obvious statistical structure related to the plaintext. We first analyzed the statistical moments of the generated feature vectors. As shown in the top subfigure of Fig. 2, regardless of the input password category (ranging from low-entropy "Simple" to high-entropy "Complex"), the hashed feature vectors exhibit a zero-centered mean with a stable standard deviation. This indicates that TSNC successfully normalizes the feature space, preventing statistical bias attacks. The near-zero mean and stable standard deviation suggest feature whitening and reduced category-dependent bias in the tested samples.

To further visualize the high-dimensional manifold of the hashed features, we employed t-Distributed Stochastic Neighbor Embedding (t-SNE). As illustrated in the bottom subfigure of Fig. 2, the feature vectors from different categories are uniformly scattered and inextricably mixed in the 2D projection. Unlike classification tasks where cluster separability is desired, the absence of discernible clusters here is a strong indicator of security. It confirms that TSNC destroys the semantic structure of the original passwords, reducing observable structure in the output features.

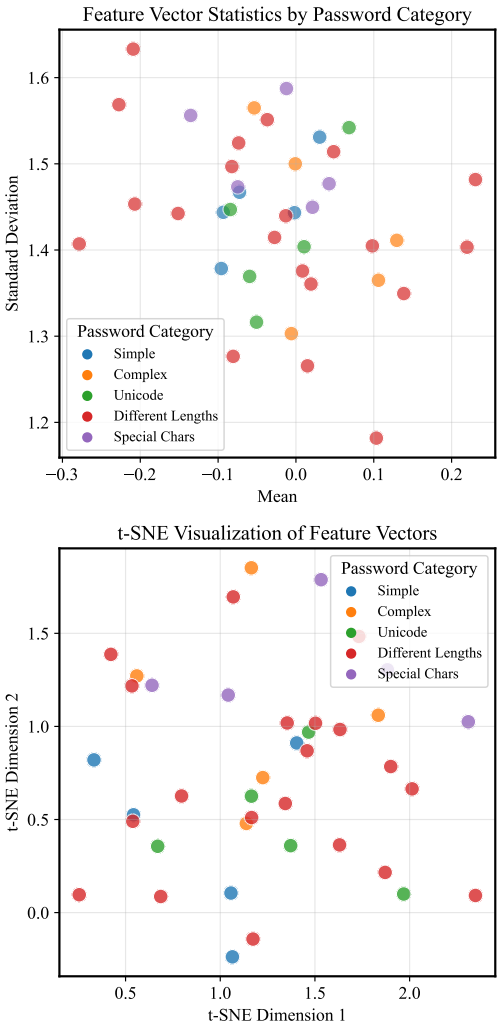

Figure 2: Feature distribution analysis. Top: Statistical moments across categories. Bottom: t-SNE visualization showing the uniform mixing of feature vectors, indicating high entropy and resistance to pattern recognition.

4.3 Temporal-stochastic orthogonality

The core innovation of TSNC lies in its temporal-stochastic mechanism, where the account creation timestamp acts as a dynamic, non-reproducible seed. We define this property as "Temporal Orthogonality"—the ability to generate statistically independent representations for the identical password under different temporal contexts. We first examined a standard password ("password123") across varying timestamps. Fig. 3 visualizes the t-SNE projection, where the distinct spatial separation between points confirms that temporal variations induce a global shift in the feature space. This observation is quantitatively supported by the cosine similarity matrix in Fig. 4. The pairwise similarities between hashed representations of the same password consistently remain below 0.2, demonstrating that the timestamp effectively acts as a "dynamic salt", rendering precomputed rainbow tables obsolete.

To rigorously test the lower bound of this security mechanism, we analyzed the simplest possible input: the single-digit password "1". In conventional hashing, such low-entropy inputs are the most vulnerable. However, Fig. 5 reveals that even for this trivial input, minimal timestamp variations trigger drastically different activation patterns in the first 32 dimensions. This divergence is further corroborated by Fig. 6, which plots the feature value distributions for these instances. As observed, each timestamp condition yields a distinct statistical footprint, effectively ensuring that the feature space is reshaped rather than merely shifted. Finally, the cross-timestamp similarity matrix in Fig. 7 shows near-zero off-diagonal values. This confirms that TSNC transforms even this low-entropy password into a complex, high-dimensional representation that is orthogonal across the temporal domain.

4.4 Cryptographic sensitivity and collision Resistance

Building upon the statistical and temporal properties, we further evaluate the algorithm's "Avalanche Effect"—a strict criterion where a slight change in input must result in a drastic, unpredictable change in output. We evaluated sensitivity using similar password pairs with minimal edit distances (e.g., "secure" vs. "secrue"). As shown in Fig. 8, these pairs exhibit high Euclidean distances (> 20) and near-zero cosine similarities (< 0.1), comparable to the relationship between two completely unrelated random strings. Moreover, Fig. 9 details the position sensitiv-

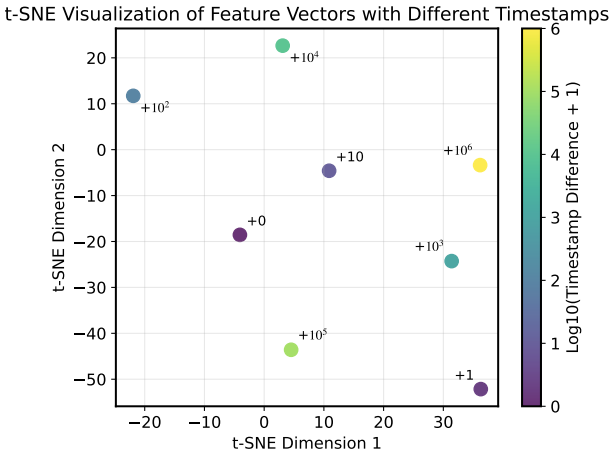

Figure 3: t-SNE visualization of feature vectors for "password123" generated under different timestamp seeds, demonstrating spatial separation.

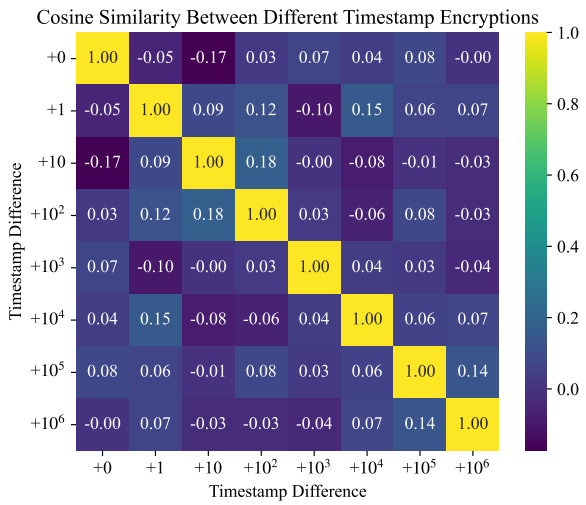

Figure 4: Cosine similarity matrix for "password123" across different timestamps. Low similarity scores (<0.2) indicate orthogonality.

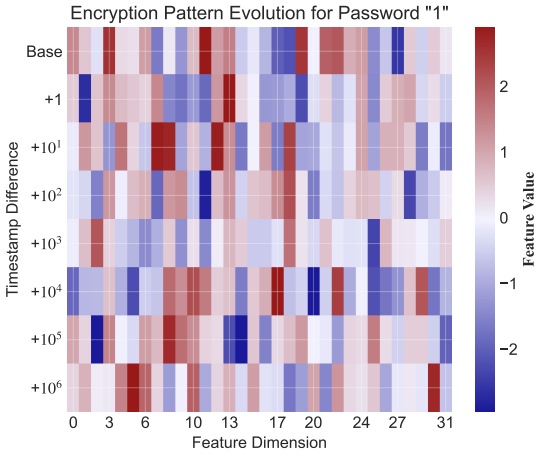

Figure 5: Heatmap of the first 32 feature dimensions for password "1" under varying timestamp differences, showing non-linear pattern evolution.

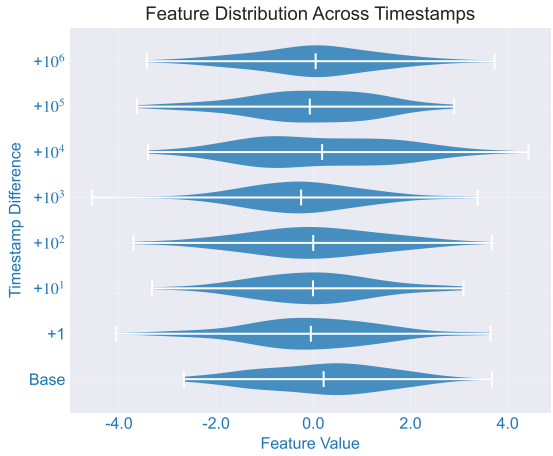

Figure 6: Distribution of feature values for password "1" across timestamps, indicating distinct statistical properties for each temporal instance.

ity. Modifying a character at any position triggers a global transformation of the feature vector. This proves that the sequential feature extraction module effectively diffuses local changes across the entire high-dimensional space.

Theoretical sensitivity implies practical collision resistance. To empirically validate this, we conducted extensive collision tests involving over 10^7 trials. We quantized the floating-point feature vectors to impose a stricter collision condition. The tests covered two scenarios: (1) Fixed Timestamp, Variable Passwords (5×10^6 trials), and (2) Fixed Password, Variable Timestamps (5×10^6 trials). Zero collisions were detected in all trials. This empirical evidence demonstrates that TSNC maintains robust injectivity, satisfying the stringent requirements of large-scale authentication systems.

Furthermore, we executed a targeted differential stress test on 10^5 password pairs with single-character variations (minimal Hamming distance) under fixed timestamps. Zero collisions were detected. This confirms that the framework's non-linear diffusion layers successfully amplify microscopic input perturbations into macroscopic output divergences, effectively thwarting differential cryptanalysis.

4.5 Timing side-channel resistance

Finally, we address the operational security of the algorithm. In real-world deployments, hashing latency can leak information about password length or internal state. Fig. 10 compares TSNC with bcrypt, PBKDF2, scrypt, and Argon2. TSNC maintains approximately 200 ms la-

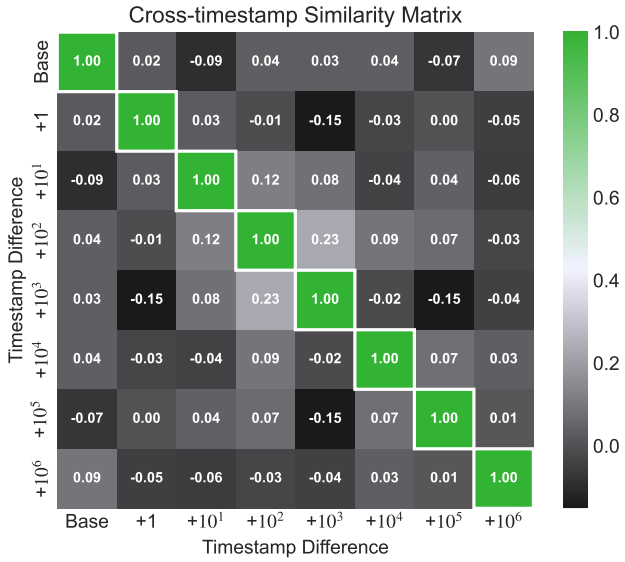

Figure 7: Cross-cosine similarity matrix for password "1". The orthogonality confirms that temporal context acts as a dynamic salt.

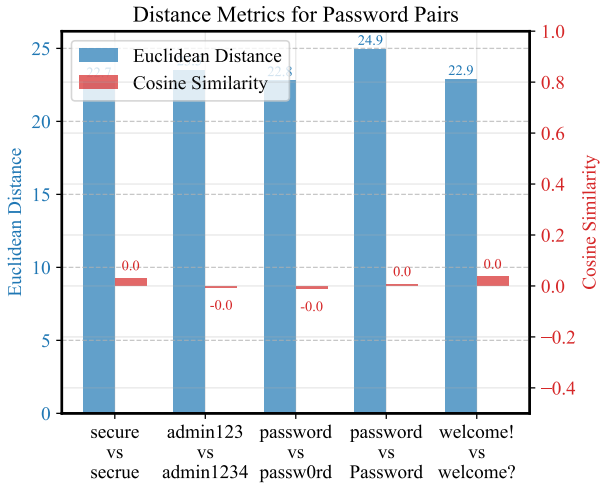

Figure 8: Sensitivity analysis for similar password pairs. High Euclidean distances and low cosine similarities confirm a strong avalanche effect.

tency across password lengths from 1 to 1000 characters, with a standard deviation below 5 ms over 50 repeated runs per length. In contrast, scrypt and Argon2 show monotonically increasing or configuration-dependent timing profiles, and Argon2d involves data-dependent memory access patterns [6]. This timing-stabilized behavior is achieved by fixed-length preprocessing and computational padding. Specifically, TSNC executes dummy matrix operations to bridge the gap between the actual workload and the upper bound determined by $M_{\max}$ and $N_{\max}$, reducing timing variation compared with data-dependent designs such as Argon2d [6].

Fig. 11 compares peak memory consumption across TSNC, PBKDF2, bcrypt, scrypt, and Argon2 for password lengths from 1 to 1000 characters. Memory use shows little dependence on password length and is mainly determined by algorithmic design. PBKDF2 and bcrypt require negligible memory, whereas scrypt and Argon2 are memory-hard and consume more memory according to their configurations [6, 24]. This increases parallel-attack cost but may limit deployment in memory-constrained environments [4]. TSNC maintains an intermediate footprint bounded by $N_{\max}$ and $M_{\max}$, balancing timing stability and memory cost.

In terms of collision resistance, conventional schemes inherit the resistance of their underlying cryptographic primitives [25, 6], consistent with TSNC's zero observed collisions over 10^7 trials in Section 4.4.

5 Conclusion

This paper proposes the Temporal-Stochastic Neural Cipher (TSNC), a training-free password hashing framework that uses account creation timestamps to initialize stochastic neural transformations. By combining timestamp-driven seed generation, sequential feature extraction, and input-dependent stochastic blocks, TSNC produces decorrelated authentication vectors for identical passwords under different temporal contexts, reducing the practicality of cross-account rainbow-table and precomputation attacks. Empirical results show zero observed collisions, whitened feature statistics, strong avalanche behavior, timing-stabilized execution, and an intermediate memory footprint. These results indicate that TSNC is suitable for authentication scenarios requiring a balance among latency, memory cost, and resistance to precomputation threats.

References

[1] Amjed A Ahmed, Mohammad Kamrul Hasan, Azana H Aman, Nurhizam Safie, Shayla Islam, Fatima A Ahmed, Thowiba E Ahmed, Bishwajeet Pandey, and Leila Rzayeva. Review on hybrid deep learning models for enhancing encryption techniques against side channel attacks. *IEEE Access*, 12:188435–188453, 2024. DOI:10.1109/ACCESS.2024.3431218.

[2] Furkan Alaca and Paul C Van Oorschot. Device fingerprinting for augmenting web authentication: classification and analysis of methods. In *Proceedings of the 32nd annual conference on computer security applications*, pages 289–301, 2016. DOI:10.1145/2991079.2991091.

[3] Yaser Alkurdi, Mustafa Al Fayoumi, Amer Albadarneh, and Qasem Abu Al-Haija. Enhancing password security via supervised learning.

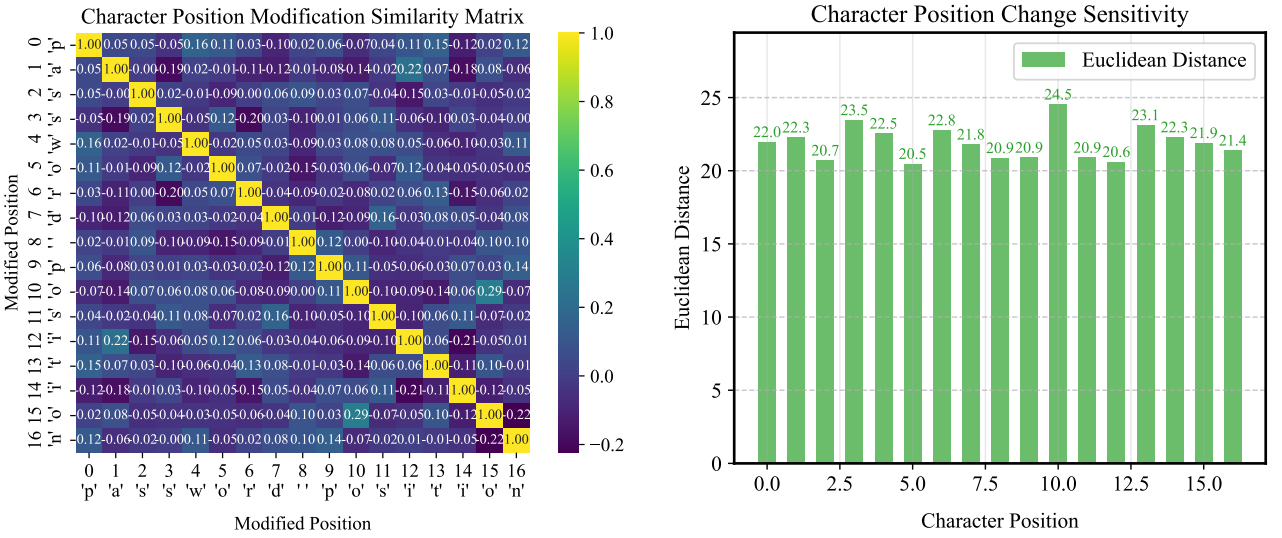

Figure 9: Character position sensitivity. Left: Euclidean distance upon single-character modification. Right: Similarity matrix showing global decorrelation.

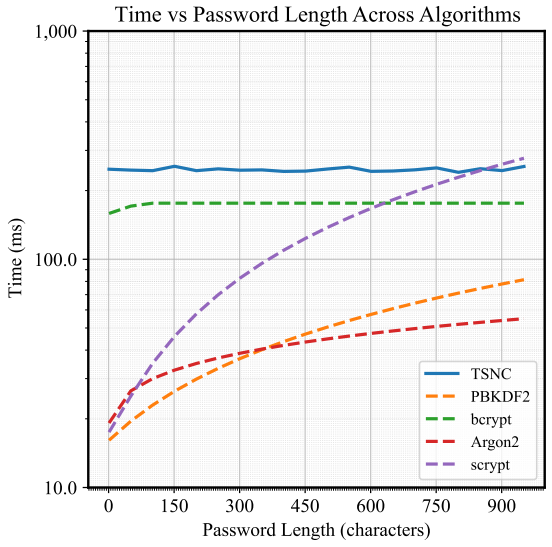

Figure 10: Hashing timing profile vs. password length. TSNC shows a timing-stabilized profile under the tested configuration, supporting its resistance to length-dependent timing leakage.

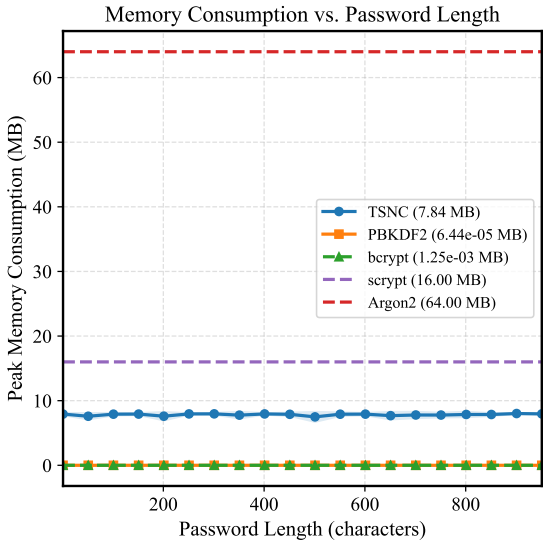

Figure 11: Peak memory consumption of TSNC and four conventional password hashing schemes (PBKDF2, bcrypt, scrypt, and Argon2) across varying password lengths. Shaded regions denote ±1 standard deviation over 10 passwords per length.

In *2023 International Conference on Information Technology (ICIT)*, pages 256–259, 2023. DOI:10.1109/ICIT58056.2023.10226141.

[4] Joël Alwen, Jeremiah Blocki, and Ben Harsha. Practical graphs for optimal side-channel resistant memory-hard functions. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1001–1017, 2017. DOI:10.1145/3133956.3134031.

[5] S.M. Bellovin and M. Merritt. Encrypted key exchange: password-based protocols secure against dictionary attacks. In *Proceedings 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 72–84, 1992. DOI:10.1109/RISP.1992.213269.

[6] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: new generation of memory-hard functions for password hashing and other applica-

tions. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 292–302. IEEE, 2016. DOI:10.1109/EuroSP.2016.31.

[7] Joseph Bonneau, Cormac Herley, Paul C Van Oorschot, and Frank Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *2012 IEEE symposium on security and privacy*, pages 553–567. IEEE, 2012. DOI:10.1109/SP.2012.44.

[8] Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. Extracting training data from large language models. In *30th USENIX security symposium (USENIX Security 21)*, pages 2633–2650, 2021. DOI:10.48550/arXiv.2012.07805.

[9] Lily Chen, Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray A Perlner, and Daniel Smith-Tone. *Report on post-quantum cryptography*, volume 12. US Department of Commerce, National Institute of Standards and Technology, 2016. DOI:10.6028/NIST.IR.8105.

[10] Angelo Ciaramella, Paolo D'Arco, Alfredo De Santis, Clemente Galdi, and Roberto Tagliaferri. Neural network techniques for proactive password checking. *IEEE Transactions on Dependable and Secure Computing*, 3(4):327–339, 2006. DOI:10.1109/TDSC.2006.53.

[11] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[12] Martin Hellman. A cryptanalytic time-memory trade-off. *IEEE transactions on Information Theory*, 26(4):401–406, 2003. DOI:10.1109/tit.1980.1056220.

[13] Burt Kaliski. Pkcs# 5: Password-based cryptography specification version 2.0. Technical report, 2000. DOI:10.17487/RFC2898.

[14] S Kayalvili, T Devadharshini, N Dharaneesh, and P Dhanush. Password strength checker using machine learning. In *2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pages 1–5, 2024. DOI:10.1109/ICCCNT61001.2024.10725352.

[15] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual international cryptology conference*, pages 388–397. Springer, 1999. DOI:10.1007/978-0-387-38162-6_6.

[16] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology—CRYPTO'96*, volume 1109, pages 104–113, Berlin, Heidelberg, 1996. Springer. DOI:10.1007/3-540-68697-5_9.

[17] Chao Liu, Zhongxiang Zheng, Keting Jia, and Qidi You. Provably secure three-party password-based authenticated key exchange from rlwe. In *International conference on information security practice and experience*, pages 56–72. Springer, 2019.

[18] Ishak Meraouche, Sabyasachi Dutta, Haowen Tan, and Kouichi Sakurai. Neural networks-based cryptography: A survey. *IEEE Access*, 9:124727–124740, 2021. DOI:10.1109/ACCESS.2021.3109635.

[19] Kathleen Moriarty, Burt Kaliski, and Andreas Rusch. Pkcs# 5: Password-based cryptography specification version 2.1. Technical report, 2017. DOI:10.17487/RFC8018.

[20] Uma Narayanan, Varghese Paul, and Shelbi Joseph. A novel system architecture for secure authentication and data sharing in cloud enabled big data environment. *Journal of King Saud University-Computer and Information Sciences*, 34(6):3121–3135, 2022. DOI:10.1016/j.jksuci.2020.05.005.

[21] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In *Annual International Cryptology Conference*, pages 617–630. Springer, 2003. DOI:10.1007/978-3-540-45146-4_36.

[22] Melissa E O'neill. Pcg: A family of simple fast space-efficient statistically good algorithms for random number generation. *ACM Transactions on Mathematical Software*, 204:1–46, 2014.

[23] François Panneton, Pierre L'ecuyer, and Makoto Matsumoto. Improved long-period generators based on linear recurrences modulo 2. *ACM Transactions on Mathematical Software (TOMS)*, 32(1):1–16, 2006. DOI:10.1145/1132973.1132974.

[24] Colin Percival and Simon Josefsson. The scrypt password-based key derivation function. Technical report, 2016. DOI:10.17487/RFC7914.

[25] Niels Provos and David Mazieres. A future-adaptable password scheme. In *USENIX annual technical conference, FREENIX track*, volume 1999, pages 81–91, 1999.

[26] Zhuoxue Song, Ziming Zhao, Fan Zhang, Gang Xiong, Guang Cheng, Xinjie Zhao, Shize Guo, and Binbin Chen. I ^{2} rnn: An incremental and interpretable recurrent neural network for encrypted traffic classification. *IEEE Transactions on Dependable and Secure Computing*, pages 1–14, 2023. DOI:10.1109/TDSC.2023.3245411.

[27] Fasee Ullah and Chi-Man Pun. Deep self-learning based dynamic secret key generation for novel secure and efficient hashing algorithm. *Information Sciences*, 629:488–501, 2023. DOI:10.1016/j.ins.2023.02.007.

[28] Ding Wang, Zijian Zhang, Ping Wang, Jeff Yan, and Xinyi Huang. Targeted online password guessing: An underestimated threat. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1242–1254, 2016. DOI:10.1145/2976749.2978339.

[29] Jingdong Wang, Ting Zhang, Nicu Sebe, Heng Tao Shen, et al. A survey on learning to hash. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):769–790, 2017. DOI:10.1109/TPAMI.2017.2699960.

[30] Zhihua Wang, Yu Liu, Jiarui Wang, Zeminghui Li, Zhenyu Li, Xiaolong Yang, Fazhi Qi, and Hongyong Jia. A reliable physical layer key generation scheme based on rss and lstm network in vanet. *IEEE Internet of Things Journal*, 11(1):692–707, 2024. DOI:10.1109/JIOT.2023.3286120.

[31] Ziming Zhao, Zhaoxuan Li, Jialun Jiang, Fengyuan Yu, Fan Zhang, Congyuan Xu, Xinjie Zhao, Rui Zhang, and Shize Guo. Ernn: Error-resilient rnn for encrypted traffic detection towards network-induced phenomena. *IEEE Transactions on Dependable and Secure Computing*, pages 1–18, 2023. DOI:10.1109/TDSC.2023.3242134.

[32] R Abu Zitar, Nidal Al-Dmour, Mirna Nachouki, Hanan Hussain, and Farid Alzboun. Hashing generation using recurrent neural networks for text documents. *ICIC Express Letters, Part B: Applications*, 12(3):231–241, 2021.