An LLM Framework For Cryptography Over Chat Channels

Danilo Gligoroski* Mayank Raikwar† Sonu Kumar Jha‡

April 11, 2025

Abstract

Recent advancements in Large Language Models (LLMs) have transformed communication, yet their role in secure messaging remains underexplored, especially in surveillance-heavy environments. At the same time, many governments all over the world are proposing legislation to detect, backdoor, or even ban encrypted communication. That emphasizes the need for alternative ways to communicate securely and covertly over open channels. We propose a novel cryptographic embedding framework that enables covert Public Key or Symmetric Key encrypted communication over public chat channels with human-like produced texts. Some unique properties of our framework are: 1. It is LLM agnostic, i.e., it allows participants to use different local LLM models independently; 2. It is pre- or post-quantum agnostic; 3. It ensures indistinguishability from human-like chat-produced texts. Thus, it offers a viable alternative where traditional encryption is detectable and restricted.

Keywords: LLMs, Transformers, Steganography, Watermarking, Cryptography

*Norwegian University of Science and Technology, Trondheim, Norway, email: danilog@ntnu.no
†University of Oslo, Oslo, Norway, email: mayankr@ifi.uio.no
‡Norwegian University of Science and Technology, Trondheim, Norway, email: sonu.k.jha@ntnu.no

Contents

1 Introduction

It is an undisputed and consensually accepted fact that Artificial Intelligence (AI) and Machine Learning are the most disruptive technologies that cause civilization and societal transformation. They profoundly affect industry, economy, work relations, and manufacturing procedures [32].

Large Language Models (LLMs), most notably ChatGPT [27] but also free and open source models like LLaMA [34], and DeepSeek [4], recently the top-ranked Grok 3 [39] (in [9] where more than 200 models are ranked), are the key part of why AI is seen as a disruptive technology. They're particularly impactful in human language applications like customer service, education, task automation, and efficiency improvement. While other AI fields, such as computer vision, are also disruptive, LLMs are currently at the forefront due to their recent popularity and broad applications. LLM models possess remarkable capabilities in generating human-like text responses and addressing queries across various aspects of daily life. Leveraging the attention mechanism, they process and generate contextually relevant responses with high coherence. LLMs can also facilitate communication between two parties, each using their own LLM to generate responses, enabling AI-mediated conversations.

With increasing interest in LLMs, significant research [10, 40, 11, 41] has focused on embedding secret messages within LLM-generated responses, particularly for secure communication between two parties. These approaches enable private communication over public channels, as LLM responses are inherently transmitted in such settings. The security of these methods is fundamentally based on the principle that an LLM-generated response containing an embedded secret message must be indistinguishable from a regular LLM-generated response without any hidden information, preventing adversarial detection.

Nevertheless, all the previous approaches are based on certain assumptions, e.g., about the minimum entropy of the public channel or the implementation of a random oracle. These assumptions are rather strict and hard to follow while implementing these approaches. Therefore, in this work, we remove these assumptions and present a construction to enable private communication over public chat channels. To do that, we construct a special function that embeds a given set of characters in specific positions in LLM-generated text. Though there is a recent work [12] addressing if ChatGPT can count letters, the special function in our work addresses a bit similar but much harder problem of indistinguishable embedding ciphertexts at exactly desired positions. This challenge becomes more critical amid rising threats to user data privacy.

1.1 Motivation

Recent political agendas and actions significantly threaten user data privacy, as evidenced by multiple recent events. The UK government has demanded that Apple implement a backdoor to access users' encrypted data [24]. Similarly, the French government considered measures to allow message transmission within the framework of investigative requests [25]. In a related development, Russia-backed hacking groups have devised techniques to compromise encrypted messaging services, including Signal, WhatsApp, and Telegram [16]. These developments raise serious concerns about the future of secure communication. Given the potential scenario where public communication lacks encryption, it becomes crucial to explore alternative methods for embedding hidden information within publicly available content. This work addresses this challenge and proposes a novel approach to achieving covert communication under such constraints.

While existing techniques such as anamorphic encryption [31] provide a means of protecting privacy, their applicability remains limited under certain constraints. Notably, if a repressive regime were to monitor, detect, and ultimately ban all conventional encryption methods, there would be a critical need for alternative ways to communicate over open channels securely. Motivated by this challenge, our work proposes a framework to facilitate covert communication under such restrictive conditions.

1.2 Contribution

In this paper, we present a novel framework for covert encrypted communication over public chat channels. The contributions of the paper are as follows.

1. We construct a novel function EmbedderLLM that algorithmically places given characters within contextually appropriate words at specific positions in the LLM-generated response.

2. We propose a framework where we use EmbedderLLM to conduct symmetric LLM cryptography or public-key cryptography.

   (a) As a use case example, we show how a password-based authenticated encryption scheme can be implemented to use EmbedderLLM. It embeds a secret message within the produced ciphertext encoded in an LLM-generated text.

   (b) As another use case example, we show how to implement an Elliptic Curve Diffie−Hellman key exchange with Ephemeral keys (ECDHE) within our framework.

1.3 Paper Organization

The remainder of the paper is organized as follows: Section 2 reviews related work. Section 3 provides an overview of transformers. Section 4 introduces the notation and details the construction of EmbedderLLM. Section 5 explores the LLM framework to enable LLM-based cryptography, categorizing approaches into symmetric and public-key cryptography. Finally, Section 6 concludes the paper and outlines promising directions for future research.

2 Related Work

Anamorphic Encryption Persiano et al. [31] invented the notion of anamorphic encryption, which allows private communication between parties even if a dictator gets the secret keys of the parties. Technically, an anamorphic encryption scheme enables two parties, sharing a double key, to embed covert messages within ciphertexts of an established PKE scheme. This protects against a dictator who can force the receiver to disclose the PKE secret keys, but remains unaware of the existence of the double key.

Banfi et al. [3] refine the original anamorphic encryption model, identifying two key limitations. First, the original scheme restricts double key generation to once, requiring a new key-pair after a dictator takes power, which may raise suspicion. Second, the receiver cannot determine if a ciphertext contains a covert message. To address these, Banfi et al. propose a model allowing multiple double keys per public key and enabling covert channels after key deployment. They provide constructions showing schemes like ElGamal, Cramer-Shoup, and RSA-OAEP support these robust extensions, enhancing secure communication in authoritarian regimes.

Following the work of Banfi et al., another important contribution comes from Catalano et al. [8]. In this work, the authors investigate the constraints of implementing anamorphic encryption using black-box techniques, focusing on the message space size. They show that any black-box approach can only support a message space that is polynomially bounded by the security parameter, limiting its scalability. Moreover, they prove that certain stronger forms of anamorphic encryption, like asymmetric anamorphic encryption [7], cannot be achieved through black-box constructions. However, under specific assumptions about the underlying public-key encryption scheme, the authors demonstrate that it is possible to realize anamorphic encryption with a much larger message space, providing valuable insights into the practical limitations and possibilities of black-box anamorphic encryption.

While anamorphic encryption is effective in mitigating risks under a dictatorship, several challenges remain. For instance, the frequency of encrypted communications between two parties may arouse suspicion, prompting the dictator to take measures that disrupt private communication. Another potential issue arises if the dictator outright bans encrypted communications or anamorphic schemes hindering surveillance [6], forcing citizens to communicate publicly. In such a scenario, private exchanges between individuals would be entirely thwarted, rendering secure communication using anamorphic encryption or any sort of encryption impossible.

Steganography Steganography enables covert communication by embedding secret messages within carrier signals such as text, images, audio, or video. In linguistic steganography (LS), natural language text serves as the carrier, producing a steganographic (stego-) text that conceals hidden information. The primary challenge in LS is generating stego-texts that not only encode secret messages but also maintain naturalness and fluency to avoid detection.

Wang et al. [37] propose DAIRstega, a linguistic steganography method that dynamically allocates coding intervals based on token conditional probabilities using a roulette wheel approach. By favoring higher-probability tokens, DAIRstega enhances the naturalness of the generated text while embedding secret messages efficiently. This approach improves stego-text quality, making it harder for adversaries to distinguish from ordinary text.

Steganographic communication typically occurs in a public setting where an eavesdropper, Eve, attempts to detect hidden messages. Alice, the sender, must ensure that Bob, the receiver, can decode the message while minimizing the risk of Eve discovering its presence. A conventional analogy compares Bob to a prisoner receiving a letter from Alice, a family member outside the prison, while Eve, the prison guard, scrutinizes the letter for unusual content. Traditional linguistic steganography methods modify an existing cover text through subtle alterations, such as synonym replacements, to avoid detection. However, with advancements in generative models, particularly LLMs, coverless steganography has emerged as a promising alternative. This approach generates stego-texts that appear indistinguishable from natural language while embedding more information within shorter messages compared to conventional cover-text-based techniques.

Huang et al. [17] introduce a coverless LLM-based steganography method where an arithmetic coding decoder guides text generation to embed secret messages seamlessly. Their approach optimizes a modified probability distribution for token generation while imposing a KL divergence constraint to balance secrecy and fluency. Their work demonstrates how LLMs can be leveraged to improve the security and reliability of linguistic steganography. Building upon this, Huang et al. [18] introduce a framework that enhances embedding efficiency by modeling the steganographic process as a Constrained Markov Decision Process (CMDP). In recent work, Bai et al. [2] introduced a method that pseudorandomly shifts the probability interval of an LLM's distribution to create a private distribution for token sampling.

An LLM-based steganography has certain assumptions. For example, the tokenization process for the sender and the receiver parties must match. Another assumption is about the entropy of the channel which defines the undetectability property of the steganography. However, these assumptions can be exploited, e.g., by tokenization error, creating a drawback of LLM-steganography.

However, a recent work [36] employs a similar idea of embedding a message within a context using sparse sampling for provably secure steganography. Nevertheless, technicalities such as optimal ranges for human-like chat conversations, embedding of the most frequent characters, cryptographic approach for defining the framework, and broadness to cover both symmetric and public-key cryptography is where we differ.

Watermarking Watermarking, like steganography, embeds secret messages in a model's output. However, unlike steganography, watermarking ensures the message remains detectable even after modifications. With the rise of LLMs, research has increasingly focused on developing robust watermarking techniques for generated text [10, 40, 19, 11, 41, 42], as detailed in recent surveys [20, 21]. While watermarking is used to verify ownership, authorship, or track usage across various media, steganography prioritizes confidentiality and undetectability, concealing the the hidden message's presence.

Recent advancements in watermarking techniques for LLMs have focused on embedding undetectable watermarks to ensure the integrity and authenticity of generated text. Christ et al. [10] introduced a cryptographically-inspired method where watermarks can only be detected with a secret key, making it computationally infeasible for unauthorized users to distinguish between watermarked and non-watermarked outputs. This approach ensures that the quality of the text remains unaffected, and the watermark remains undetectable even under adaptive querying.

Building upon the concept of undetectable watermarks, Zamir [40] proposed a method to hide arbitrary secret payloads within LLM responses. A secret key is required to extract the payload, and without it, distinguishing between original and payload-embedded responses is provably impossible. Notably, this technique preserves the generated text quality, thereby extending the applicability of undetectable watermarking in secure communications.

To counter adaptive users, Cohen et al. [11] developed a multi-user watermarking scheme that traces model-generated text to users or groups, even under adaptive prompting. Their construction builds on undetectable, adaptively robust, zero-bit watermarking, ensuring watermark detectability despite text modifications.

To further enhance the resilience of watermarks against various text alterations, recent research [41, 42] has proposed methods that maintain the watermark's detectability even after the text undergoes paraphrasing or other modifications. These approaches aim to ensure that the embedded watermarks are not only

imperceptible but also robust against a range of potential attacks, thereby strengthening the security and reliability of watermarking in LLMs.

These studies collectively illustrate the evolving landscape of LLM watermarking techniques, emphasizing the need to balance undetectability, robustness, and adaptability to diverse user behaviors and adversarial threats. However, key properties such as robustness and publicly accessible detection APIs can also introduce vulnerabilities, potentially exposing these systems to various attacks [29, 30, 13].

3 Transformers

LLMs represent a specific type of neural network architecture. They use transformers and attention mechanisms, which Vaswani et al. introduced in a 2017 paper [35] titled "Attention Is All You Need."

The fundamental mathematical techniques and theories employed in all LLMs include: 1. Linear Algebra methods such as high-dimensional embeddings and matrix operations; 2. Probability Theory, particularly the softmax functions that transform numerical values (typically the results of dot products) into probabilities (probability distributions) over tokens; and 3. Calculus, notably partial derivatives used in gradient descent algorithms to identify minima.

In this work, we will use some of the transformer's parameters known as *temperature* T, *top-k*, and *top-p (nucleus) sampling*. Those parameters are used to control the diversity and quality of the generated text. Let us briefly elaborate on the role of these parameters.

Transformers output a set of raw scores, called *logits*, for each token in the vocabulary. These logits, denoted z_i for token i, are converted into probabilities using the *softmax function*:

$$p_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Here, p_i represents the probability of token i, and the sum is calculated across all tokens in the vocabulary. The model then samples the next token from this probability distribution. Parameters such as temperature T, top-k, and top-p adjust these probabilities.

Temperature T, is a parameter that scales the logits before applying the softmax function. The modified probability distribution becomes:

$$p_i = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}} \tag{1}$$

- When $T = 1$: This is the standard softmax, and the probabilities reflect the model's raw predictions. When $T > 1$: Dividing the logits by a larger T diminishes the magnitude of the exponents, making the distinctions between logits less emphasized. This flattens the distribution and leads to probabilities that are more uniform. Consequently, the model becomes less confident, fostering increased diversity in token selection.

- When $T < 1$: Dividing by a smaller T amplifies the exponents, exaggerating differences between logits. This sharpens the distribution, making high-probability tokens even more likely and low-probability ones less likely. The output becomes more deterministic and focused.

Thus, temperature controls the *smoothness* or *confidence* of the probability distribution: High temperature ($T > 1$) invokes more randomness and creativity; Low temperature ($T < 1$) causes more predictable, high-confidence outputs.

Top-k sampling restricts the model to sample from only the k *most likely tokens*, rather than the entire vocabulary. More concretely:

1. Identify the k tokens
$$Y_{top-k} = \{y_1, \ldots, y_k\} \tag{2}$$
with the highest logits, say $z_{i_1}, z_{i_2}, \ldots, z_{i_k}$ (where $i_1, i_2, \ldots, i_k$ are the indices of these tokens).

2. Compute probabilities only over Y_{top-k} tokens, setting the probabilities of all other tokens to zero:

$$p_{i_j} = \frac{e^{z_{i_j}}}{\sum_{m=1}^{k} e^{z_{i_m}}} \tag{3}$$

for $j = 1, 2, \ldots, k$, and $p_i = 0$ for all other tokens.

3. Sample the next token from this truncated distribution.

Thus, the effect of top-k sampling is that it eliminates unlikely tokens, which reduces the chance of incoherent or random outputs. Smaller values of k create more focused output, while larger values of k allow for greater diversity among the top candidates.

Top-p sampling, also known as nucleus sampling, is a technique that dynamically selects a subset of tokens based on cumulative probability instead of using a fixed number k. Given a threshold p (e.g., 0.9):

1. Sort the tokens by decreasing probability: $p_{i_1} \geq p_{i_2} \geq \cdots \geq p_{i_n}$.

2. Find the smallest set of tokens (the "nucleus") such that their cumulative probability is at least p:

$$\sum_{j=1}^{k} p_{i_j} \geq p$$

3. Sample only from these k tokens, either by keeping their original probabilities or renormalizing them over the nucleus (depending on the implementation).

Unlike top-k, which fixes the number of tokens, the effect of top-p is that it adapts the size of the candidate set based on the distribution. It focuses on the most probable tokens that collectively account for p of the probability mass, balancing diversity and coherence.

Temperature and sampling methods like top-k or top-p are often used together. The standard approach is:

1. Apply temperature to the logits to adjust the distribution: z_i/T.

2. Compute probabilities using the softmax function.

3. Apply top-k or top-p sampling to truncate the distribution.

In this work, when we write

$$Y_{top-k} \leftarrow \text{LLM}(\mathit{TOPIC}, \mathit{Story}, T, k) \tag{4}$$

It means that we ask LLM to extend the story *Story* within the topic *TOPIC*; moreover, instead of picking a token, we ask the transformer to use the temperature T and to identify the top k most probable tokens $Y_{top-k} = \{y_1, \ldots, y_k\}$. In the next section, we will add more criteria for selecting which token from Y_{top-k} to choose.

Since this research aims to establish a framework for strong cryptographic communication over human-like texts generated by LLMs, a generic advice for the parameters is: Start with $T = 0.7$. It will produce a focused story that still has an internal variety and will mimic a human who types in a thoughtful, yet casual way. Choose $k = 40$ for producing texts with lexical variety, yet still coherent and diverse. That resembles how humans choose words from their familiar word capacity. While the generic LLM folklore advice concerning the value p is to be in the range $[0.9, 0.95]$, and many LLM use cases that mimic human-like texts prefer using the top-p parameter over top-k, in this work, we will not use and change top-p parameter.

4 Notations and a special function EmbedderLLM

Before we outline our framework for performing cryptographic operations on LLM outputs, let us first introduce the following notations and definitions.

Let us denote the sets S_i, $i = 1, \ldots, 4$ as $S_1 = \{\texttt{0}, \texttt{1}\}$, $S_2 = \{\texttt{0}, \texttt{1}, \texttt{2}, \texttt{3}\}$, $S_3 = \{\texttt{0}, \ldots, \texttt{7}\}$, $S_4 = \{\texttt{0}, \ldots, \texttt{F}\}$, where the elements in S_4 are the 16 hexadecimal numbers from 0 up to F.

In a similar manner, we denote the sets L_i, $i = 1, \ldots, 4$ as

$$L_1 = \{\texttt{' '}, \texttt{E}\},$$
$$L_2 = \{\texttt{' '}, \texttt{E}, \texttt{T}, \texttt{A}\},$$
$$L_3 = \{\texttt{' '}, \texttt{E}, \texttt{T}, \texttt{A}, \texttt{O}, \texttt{N}, \texttt{I}, \texttt{S}\},$$
$$L_4 = \{\texttt{' '}, \texttt{E}, \texttt{T}, \texttt{A}, \texttt{O}, \texttt{N}, \texttt{I}, \texttt{S}, \texttt{R}, \texttt{H}, \texttt{D}, \texttt{L}, \texttt{U}, \texttt{C}, \texttt{M}, \texttt{F}\}.$$

We note that while $L_1 \subset L_2 \subset L_3 \subset L_4$, the set L_4 is a sorted set of 15 most frequent letters in English texts prepended with an even more frequent character in English texts, and which is the blank (SPACE) character. The letters' frequencies are taken from [38] and adjusted by adding the SPACE character as done in [1] and here presented in Table 1. The case of the letters (lowercase or uppercase) was considered equivalent. Notice the bold font probabilities for the letters E, A, S and F.

For $i = 1, \ldots, 4$, we define four bijective maps $h_i : S_i \to L_i$ where the mapping is the natural mapping between the ordered sets S_i and L_i, i.e., 0 $\mapsto$ ' ', 1 $\mapsto$ E, 2 $\mapsto$ T and so on until F $\mapsto$ F. While h_1, h_2, and h_4 are the most adequate maps for our purposes, we note that with some technical tricks (padding the ciphertexts), we can also use the map h_3.

Table 1: Frequencies of the top 16 characters in English texts (including SPACE) [1].

h	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	SPACE	E	T	A	O	N	I	S	R	H	D	L	U	C	M	F
Prob	19.18%	**10.41%**	7.29%	**6.52%**	5.96%	5.65%	5.58%	**5.16%**	4.98%	4.93%	3.50%	3.31%	2.25%	2.17%	2.02%	**1.98%**

With a slight abuse of notation, when we write $h_i(Enc)$ for a hexadecimal string Enc, we refer to a character-by-character mapping of each character in Enc to its corresponding image defined by h_i.

We will extend the generic top-k sampling described with equations (3) and (4) with the function EmbedderLLM. This function adds additional criteria for selecting the next token from Y_{top-k}. The details of this function are as follows:

$$\text{EmbedderLLM}(\text{LLM}, \mathit{TOPIC}, \mathit{Story}_0, T_0, k_0, \mathbf{C}, \mathbf{b})$$

It outputs a string $Story$. As input parameters, it receives the name of a specific LLM model to be used, a topic $TOPIC$ discussed in the $Story$, a potentially previously generated $Story_0$ that we want to continue, a starting value T_0 for a temperature, an initial value k_0 for the top k_0 tokens with top k_0 probabilities, a sequence $\mathbf{C} = [C_0, C_1, \ldots, C_{n-1}]$ of characters from some of the sets $S_1, \ldots, S_4$, and an increasing sequence of integers $\mathbf{b} = [b_0, b_1, \ldots, b_{n-1}]$. Notice that both $\mathbf{C}$ and $\mathbf{b}$ have the same number of elements n.

An implicit yet crucial parameter regarding the generation of the sequence $\mathbf{b}$ is a parameter d_o called the offset value. Its role is described as follows. Let us first denote by PRF() some pseudo-random cryptographic function that, once properly initiated, with each call, gives cryptographically strong uniformly distributed pseudo-random integer outputs in the range $[0, 2^{bit\_chunk\_size})$ for some value of bit_chunk_size. For example, if $bit\_chunk\_size = 5$, the function PRF() will give uniformly distributed outputs in the range $[0, 32)$. Next, when we write $b =$ PRF() it means that b has received a pseudo-random integer value generated by PRF() and b is uniformly distributed in the range $b \in [0, 2^{bit\_chunk\_size})$.

Then, the sequence $\mathbf{b} = [b_0, b_1, \ldots, b_{n-1}]$ is recursively generated as follows:

$$\begin{cases} b_0 & \leftarrow d_o + \texttt{PRF}(), \\ b_i & \leftarrow b_{i-1} + d_o + \texttt{PRF}(), \quad \text{for } \ i \in [1, n-1] \end{cases} \tag{5}$$

Table 2: Digram frequencies of the top 16 characters in English texts (including SPACE) [1].

	SPACE	E	T	A	O	N	I	S	R	H	D	L	U	C	M	F
SPACE	0.056	0.021	0.146	0.106	0.065	0.023	0.068	0.07	0.021	0.061	0.028	0.024	0.009	0.039	0.041	0.04
E	0.353	0.032	0.024	0.055	0.003	0.082	0.013	0.071	0.131	0.002	0.084	0.034	0.001	0.022	0.019	0.009
T	0.245	0.089	0.017	0.039	0.091	0.001	0.067	0.025	0.031	0.335	**7E-05**	0.015	0.019	0.004	9E-04	9E-04
A	0.072	0.001	0.132	3E-04	5E-04	0.215	0.037	0.093	0.11	0.001	0.043	0.078	0.01	0.037	0.026	0.008
O	0.126	0.003	0.049	0.008	0.033	0.146	0.009	0.032	0.118	0.002	0.02	0.027	0.134	0.013	0.058	0.114
N	0.257	0.085	0.096	0.024	0.065	0.009	0.03	0.043	4E-04	0.003	0.177	0.009	0.009	0.042	7E-04	0.005
I	0.056	0.048	0.12	0.017	0.05	0.262	5E-04	0.116	0.044	8E-05	0.036	0.053	7E-04	0.049	0.041	0.027
S	0.401	0.118	0.127	0.035	0.049	0.002	0.051	0.046	2E-04	0.057	0.002	0.011	0.033	0.016	0.007	0.001
R	0.224	0.235	0.042	0.073	0.092	0.022	0.079	0.046	0.019	0.003	0.028	0.012	0.017	0.012	0.019	0.006
H	0.114	0.46	0.03	0.158	0.07	0.001	0.125	0.002	0.012	**7E-05**	5E-04	0.001	0.013	0.002	0.001	5E-04
D	0.62	0.122	9E-04	0.028	0.04	0.003	0.079	0.031	0.018	7E-04	0.008	0.009	0.018	3E-04	0.002	7E-04
L	0.144	0.166	0.018	0.102	0.083	1E-03	0.111	0.023	0.002	2E-04	0.067	0.136	0.014	0.002	0.006	0.013
U	0.058	0.031	0.125	0.026	0.002	0.116	0.03	0.121	0.163	7E-05	0.022	0.107	**3E-05**	0.046	0.031	0.006
C	0.015	0.172	0.091	0.123	0.176	**4E-05**	0.057	0.001	0.038	0.17	5E-04	0.049	0.036	0.022	**0**	**9E-06**
M	0.158	0.25	8E-04	0.154	0.11	0.003	0.088	0.025	0.017	3E-04	0.003	0.002	0.039	2E-04	0.022	0.002
F	0.373	0.093	0.034	0.064	0.155	9E-05	0.082	0.004	0.083	1E-04	2E-04	0.027	0.033	3E-04	**4E-05**	0.05

The goal in defining EmbedderLLM is to produce a grammatically correct and sound string *Story* such that its length is constrained

$$b_{n-1} + 1 \leq \text{len}(Story) \leq b_{n-1} + d_o - 1,$$

and where the set $\mathbf{C}$ is embedded in *Story* exactly on positions given in $\mathbf{b}$, i.e.,

$$\text{Uppercase}(Story[b_i]) = C_i, \text{ for } i \in [0, n),$$

and the words containing characters C_i are from language dictionary on which LLM was trained. In our experiments, we used LLMs trained in the English language.

Before we describe algorithmically the function EmbedderLLM, let us introduce the following additional notations.

We overload the definition of indexing the characters of a string as follows:

$$\texttt{Char}(string, b) = \begin{cases} string[b], & \text{if } b < len(string) \\ \textbf{None}, & \text{if } b \geq len(string) \end{cases} \tag{6}$$

where $len(string)$ returns the length of *string* and $string[b]$ is the usual 0-based Python string indexing. The reason for this overloading is that the usual Python string indexing will return `IndexError : string index out of range` if the value of b is greater or equal to the length of the string.

When we write

$$string \leftarrow \varepsilon,$$

then means *string* becomes an empty string.

The expression

$$string \leftarrow string||token,$$

means that the new value of *string* is the old *string* concatenated with another string called *token*, while the expression

$$string \leftarrow string[: position],$$

means a classical Python string slicing such that the new value of *string* is the old *string* stripped off by the characters from the index *position*.

If we write

$$token \xleftarrow{\$} Y,$$

it means that *token* gets a value from the set of tokens Y and *token* is chosen uniformly at random from Y. Finally the expression

$$A_{\texttt{shuffled}} \leftarrow \mathsf{RandomShuffle}(A),$$

means the elements in $A_{\texttt{shuffled}}$ are randomly reordered elements from A.

The crucial algorithm in our framework for LLM Cryptography is Algorithm 1. Let us discuss its steps. After the initialization steps 1 and 2, it reads a value from Table 3 for the maximum number of repetitive attempts to find an appropriate token by increasing just the temperature before it rises the k parameter. In Step 4 it calculates the temperature increase amount. Then it enters the main 'while' loop for updating a human-like *Story* about the topic *TOPIC*, that contains embedded characters $\mathbf{C} = [C_0, C_1, \ldots, C_{n-1}]$ on positions $\mathbf{b} = [b_0, b_1, \ldots, b_{n-1}]$. The 'while' loop is executed as long as the counter i is less than n. In Step 6 it calls the LLM model to return a list of k tokens that are candidates for continuing the *Story*. Then, in Step 7, it checks whether there are tokens $\{y\}$ that, if appended to the *Story*, would contain the character C_i exactly at position b_i. The set of such tokens is named Y. In Step 8 it checks if Y is a non-empty set. If yes, then in Steps 9 – 15, it chooses uniformly at random an element of Y, it updates *Story* by appending the *next_token*, it remembers the position in *Story* where the last successful embedding happened, it resets the values of temperature T, k, *Slow_Down*, and *Close* to their initial starting values (since it might happen later in the algorithm that we updated these parameters) and increases the counter i. If the set Y is empty, the algorithm tries to append an in-between token from the initial set Y_{top-k} that is randomly shuffled in Step 17. It tries all tokens from the shuffled list (Steps 19 – 41) and checks for the first case where $len(Story||next\_token) < b_i - 6$. If that happens, it updates *Story* with a new token (Step 21). The reason why we put the limiting value $b_i - 6$ and not b_i is that we want to detect the event when the length of *Story* will approach the crucial position b_i. Once entering the proximity of b_i we want to try up to *top_f* attempts to find a non-empty Y by just increasing the temperature T in Step 34, before go to the part of the algorithm in Step 42. The algorithm would reach Step 42 if there was no appropriate token in all attempted Y_{top-k}. In that case, the algorithm shortens the story to the length where the last successful embedding happened (Step 43), relaxes both parameters T and k (Steps 44 and 45), and resets the variables *Slow_Down* and *Close*. In such a case, the 'while' loop continues with the relaxed parameters and tries again to embed C_i at the position b_i.

First, let us justify using a non-zero value d_o in (5). Let the sequences $\mathbf{C} = [C_0, C_1, \ldots, C_{n-1}]$ and $\mathbf{b} = [b_0, b_1, \ldots, b_{n-1}]$ be the sequences produced as described above, where `PRF`() in the recurrent equations (5) gives uniformly distributed integers in the range $b \in [0, 2^{bit\_chunk\_size})$.

Table 2 gives the probability of diagrams in English texts. Apparently, there are digrams with zero or very low probability. We can look at digrams as a pair of characters with an offset $d_o = 0$. Then, in a sufficiently long sequence of characters, we might have the situation where (C_i, C_{i+1}) is a diagram with zero or very low probability. From Table 2 such digrams could be `CM`, `CF`, `CN`, `TD`, `HH` and so on (presented in bold font in the table). It means that regardless of how many calls to LLM$(TOPIC, Story, T, k)$ we make in Step 5 of Algorithm 1, there will be no tokens that give non-empty set Y in Step 6. From an empirical perspective, in our experiments, we noticed that offset digrams where $d_o < 6$ still give potentially small probabilities. Thus, we set the default value $d_o = 32$.

4.1 Modeling the Distinguishability of Texts Outputs From Algorithm 1 From Human-Like Produced Texts

We are going to model the distinguishability of texts generated by Algorithm 1 from human-like produced texts within the following plausible constraints (*assumptions*):

1. The average token size [28] (in number of characters) used in modern LLMs is 4.
2. Optimal parameters for the imitation of human-produced chat texts are $T \in [0.7, 0.9]$ and $k \in [40, 60]$.

Algorithm 1 EMBEDDERLLM(LLM, $TOPIC, Story_0, T_0, k_0, \mathbf{C}, \mathbf{b}, l, sec$)

Input: LLM model, $TOPIC$ discussed in $Story$, initial content of story $Sorty_0$, initial values for temperature T_0 and k_0 for selection of top k tokens, sequence of characters $\mathbf{C} = [C_0, \ldots, C_{n-1}]$ and the sequence of integers $\mathbf{b} = [b_0, \ldots, b_{n-1}]$. The value of l can be one from $[1, 2, 3, 4]$ and determines which set L_l we use, and $sec \in [32, 48, 64, 96, 128]$ where 2^{-sec} is the probability a used token to be produced with parameters outside the optimal ranges.

Output: A text string $Story$

$i \leftarrow 0$, $n \leftarrow len(\mathbf{C})$, $Story \leftarrow Story_0$, $prev\_pos \leftarrow len(Story)$
$Close \leftarrow$ `False`, $T \leftarrow T_0$, $k \leftarrow k_0$, $Slow\_Down \leftarrow 0$
$top\_f \leftarrow Table[sec, l]$
$t_{\texttt{slow\_down}} \leftarrow \frac{0.2}{21 \times top\_f}$
while $i < n$ **do**
    $Y_{top-k} \leftarrow \text{LLM}(TOPIC, Story, T, k)$
    $Y = \{y \in Y_{top-k} \mid \texttt{Char}(Story||y, b_i) = C_i\}$
    if $Y \neq \emptyset$ **then** ▷ Successful embedding
        $next\_token \xleftarrow{\$} Y$
        $Story \leftarrow Story||next\_token$
        $prev\_pos \leftarrow len(Story)$
        $T \leftarrow T_0$, $k \leftarrow k_0$ ▷ Reset values
        $Slow\_Down \leftarrow 0$
        $Close \leftarrow$ `False`
        $i \leftarrow i + 1$ ▷ Go for the next embedding
    else ▷ Check if we can add an in-between token
        $Y_{\texttt{shuffled}} \leftarrow \mathsf{RandomShuffle}(Y_{top-k})$
        $Unsuccessful \leftarrow$ `True`
        for $next\_token \in Y_{\texttt{shuffled}}$ **do**
            if $len(Story||next\_token) < b_i - 6$ **then**
                $Story \leftarrow Story||next\_token$
                $Unsuccessful \leftarrow$ `False`
                `Break`
            else if $len(Story||next\_token) < b_i$ **then**
                if (**Not** $Close$) **then**
                    $Close \leftarrow$ `True`
                    $Story \leftarrow Story||next\_token$
                    $Unsuccessful \leftarrow$ `False`
                    `Break`
                else
                    $Slow\_Down \leftarrow Slow\_Down + 1$
                    if $Slow\_Down < top\_f$ **then**
                        $Unsuccessful \leftarrow$ `False`
                        $T \leftarrow T + t_{\texttt{slow\_down}}$ ▷ Try again. Just increase T
                        `Break`
                    else
                        $Slow\_Down \leftarrow 0$
                    end if
                end if
            end if
        end for
        if $Unsuccessful$ **then**
            $Story \leftarrow Story[: prev\_pos]$
            $T \leftarrow T + t_{\texttt{slow\_down}}$ ▷ Try again with increased T
            $k \leftarrow k + 1$ ▷ and increased k
            $Slow\_Down \leftarrow 0$, $Close \leftarrow$ `False`
        end if
    end if
end while
Return $Story$

3. We assume that the adversary possesses the exact local LLM used in the call of Algorithm 1. By this, we assume that the adversary knows even the local fine-tunings of a publicly available LLM model. This makes the adversary a very powerful entity, which in practice might not be the case, but from a security perspective, it is a preferred modeling style.

4. The adversary does not know the values of the sequences **C** and **b**.

5. Knowing the exact LLM, the adversary successfully distinguishes the captured *Story* if it distinguishes a token in *Story* that was produced with parameters T and k outside of the ranges $[0.7, 0.9]$ and $[40, 60]$ (we call these ranges "optimal ranges").

Table 3: A table of top-f parameters for achieving probabilities less than 2^{-sec} for a used token to be produced with parameters outside the optimal ranges.

		E	A	S	F
	p_l	**0.1041442**	**0.0651738**	**0.0515760**	**0.0197881**
		Table			
	l	1	2	3	4
16		17	25	29	60
32		34	49	58	120
48		51	73	87	179
64		68	97	116	239
96		102	146	174	358
128		136	194	232	477

sec

Theorem 1. *The probability a Story generated by* EMBEDDERLLM(LLM, $TOPIC, \varepsilon, T_0, k_0, \mathbf{C}, \mathbf{b}, l, sec$) *to have a token produced with parameters* $T \notin [0.7, 0.9]$ *and* $k \notin [40, 60]$ *denoted as* P_{dist} *is less than* 2^{-sec}.

Proof. Let us compute the probability of Algorithm 1 to reach and execute steps 44 – 46. First, let us use the fact that the average token size [28] in modern LLMs is four characters. Plausible modeling of k tokens whose average size is four is a multinomial distribution with a central value of length 4. More concretely, we can model that k tokens belong to seven categories (seven bins) determined by token length. We consider the probability of having a token of length eight or higher to be very small. In that case, the probability that a token has a length $j \in [1, \ldots, 7]$ can be modeled with

$$P(j) = 2^{-6}\binom{6}{j-1}, \quad j \in [1, \ldots, 7]. \tag{7}$$

Then, in Step 7, we compute the set
$Y = \{y \in Y_{top-k} \mid \texttt{Char}(Story||y, b_i) = C_i\}$. The lower bound of the probability the set Y is non-empty depends on the number k of elements in the set Y_{top-k} and the probability p_l of character with the lowest frequency in the set $L_l, l \in [1, \ldots, 4]$ (given in Table 3). The lower bound can be computed as:

$$\Pr(Y \neq \emptyset) = P(k, p_l) = \frac{1}{7}\sum_{j=1}^{7}\left(\frac{p_l}{j}\right)^{kP(j)}. \tag{8}$$

Now we can compute the upper bound of the probability of a token to be produced with parameters T and k that are outside the ranges $[0.7, 0.9]$ and $[40, 60]$ as a consecutive product of the probabilities of the opposite event, i.e.,

$$\Pr(Y = \emptyset) = 1 - \Pr(Y \neq \emptyset). \tag{9}$$

However, here, with a slight increase in the temperature (between the events when we also increase the value of k) in Step 34, we want to increase the number of attempts to find a non-empty set Y. The number of

such attempts is also given in Table 3, and plays a role in Step 4, where we calculate the appropriate value for temperature increase $t_{\texttt{slow\_down}}$. Then, the upper bound of the probability P_{dist} can be calculated with the following formula:

$$P_{dist} \leq \prod_{k=40}^{60} (1 - P(k, p_l))^{top\_f} . \tag{10}$$

For concrete values of l, i.e., p_l one can compute that indeed for the values top_f given in Table 3, the probabilities P_{dist} are upper bounded by 2^{-sec}. □

4.2 Discussion on Algorithm 1

4.2.1 Execution (in)efficiency

Algorithm 1 is not an efficient algorithm. It repeatedly calls the LLM model with slightly changed parameters in an effort to build a story with the required properties. A smarter strategy might involve the use of dictionaries, thesaurus, and grammar tools in order to replace certain words with appropriate synonyms or similar phrases that would not change the narrative and meaning but would tweak the length of the story such that the required character C_i is placed exactly on position b_i. However, in that case, the probabilities modeling that we have conducted in Theorem 1 might not hold.

4.2.2 Redundancy (in)efficiency

From the redundancy perspective, apparently, for sending a short encrypted text, Algorithm 1, embeds it in a longer *Story*. Choosing bigger values for d_o linearly increases the length of produced *Story*. Our default recommended value is $d_o = 32$. It allows every token that contains a mapped ciphertext character C_i to be at a safe distance from each other such that finding appropriate tokens is a procedure independent of the embedding token found previously. Another crucial parameter that affects the redundancy, i.e., the size of *Story*, is the parameter l, i.e., the mapping set L_1, L_2, L_3, or L_4. With smaller l, we get bigger values of n; thus, the length of *Story* increases.

4.2.3 Practical vs cryptographically strong probabilities

Table 3 displays values of top_f for different combinations of sec and l. However, from a cryptographic point of view, only the lowest row with $sec = 128$ offers parameters that ensure the probabilities are lower than 2^{-128} for producing a token with temperature T and k outside the optimal ranges. Why do we give parameters for lower probabilities? The answer can be summarized as: in practice, users might choose relaxed parameters and still be comfortable that all produced tokens, with overwhelming probabilities, will be within the optimal ranges. For example, in Steganography for distinguishing between classes, so-called Linear Distinguishing Analysis (LDA) is used, and typically, the values of ΔLDA (considered as good indistinguishing properties) are in the range $[0.004, 0.007]$ (see for example [37]). In Table 3, the parameters for $sec = 16$ already surpass that range because $2^{-16} \approx 0.000015$.

4.2.4 (In)Plausability of assumption 3

In assumption 3, we assume that the adversary possesses the exact local LLM used in the call of Algorithm 1. In practice, this makes the adversary a very strong adversary. For example, currently, on "Hugging Face" there are 1,514,210 models [15]. Moreover, users can locally produce their own variants of the models, and those models can be kept private.

4.2.5 Using Algorithm 1 with more than one LLM model

Closely related to the previous point, users can modify the algorithm so that in Step 6, it calls different LLM models. Users can even call several different models in parallel, check which one gives non-empty Y, and continue to the next C_i and b_i.

5 LLM Cryptography

5.1 Symmetric Key LLM Cryptography (Password-Based AEAD)

Key Idea: Alice and Bob want to have a secure communication. They share an initial secret password. They generate keys dk_1, dk_2 using a Password-based Key Derivation Function (PBKDF2) [26]. To send a secret message $M = plaintext$, Alice encrypts it using Authenticated Encryption with Associated Data (AEAD) function [5] with key dk_1, generating ciphertext Enc. She then maps Enc to an encoded form C using a function h, ensuring the characters belong to a set of frequent English letters. Alice embeds C in a generated *Story* on a *Topic* using EmbedderLLM, which algorithmically places each character of C within contextually appropriate words at specific positions, determined using dk_2 and public parameters. The crucial challenge is ensuring these insertions maintain the natural distribution of human-like but LLM-generated text. Alice then transmits the story over a public channel to Bob. Upon receiving it, Bob extracts C using dk_2, reverses the mapping via h^{-1}, and decrypts the recovered ciphertext using AEAD with dk_1 to retrieve M.

Construction:

Let us denote a generic Authenticated Encryption with Associated Data (AEAD) [5] function as

$$ciphertext, tag \leftarrow \text{AEAD}_{enc}(dk_1, nonce, AD, plaintext),$$

and its corresponding inverse function of decryption and verification as

$$plaintext \leftarrow \text{AEAD}_{dec}(dk_1, nonce, AD, ciphertext, tag).$$

The $\text{AEAD}_{enc}()$ function receives as parameters a secret key dk_1, publicly known values of *nonce* (which should be some non-repeatable value) and associated data AD, and a *plaintext* to be encrypted. It returns the encrypted output *ciphertext* and a verification (checksum) value *tag*. The $\text{AEAD}_{dec}()$ might return the *plaintext* if the verification tag *tag* passes the test, or it might return the value $Fail$ otherwise.

Let us further denote a generic Password Based Key Derivation Function (PBKDF) (version 2) [26] as

$$DK = \text{PBKDF2}(password, Salt, count, dkLen).$$

The PBKDF2() function receives as parameters a value of *password*, a string known as $Salt$, the number *count* of applications of some cryptographic hash function (not explicitly mentioned here, but in practical implementations should be instantiated with functions such as SHA2 or SHA3), and the size of the output in bytes as a number $dkLen$. For example, if we put $dkLen = 64$, the output DK will be long $64 \times 8 = 512$ bits.

The assumption is that Alice and Bob share a secret *password* along with other information that need not be secret. That non-secret information is:

1. Which character mapping function $h_1, \ldots, h_4$ they will use; In the descriptions below we assume they use h_4.
2. Values of *count* and $Salt$ for PBKDF2;
3. Values of *nonce* and AD for AEAD() functions;
4. An offset value d_o. As a default value, we set $d_o = 32$.
5. The authentication tag size in AEAD() functions is fixed to 128 bits, i.e., to 32 hexadecimal values.
6. The use of SHAKE128 Extendable-Output Function
7. Parsing the output of SHAKE128 into chunks of $chunk\_size$ bits. As a default value, we set $chunk\_size = 5$.

Algorithm 2 LLM Authenticated Encryption

Input: A shared secret *password*
Output: A text string *Story*

1: $dk_1, dk_2 \leftarrow \text{PBKDF2}(password, Salt, count, 64)$ ▷ assert $\text{len}(dk_1) = \text{len}(dk_2) = 256$ bits
2: Alice generates a message *plaintext* to be encrypted.
3: $ciphertext, tag \leftarrow \text{AEAD}_{enc}(dk_1, nonce, AD, plaintext)$
4: Treat *ciphertext* and *tag* as hexadecimal strings.
5: $Enc \leftarrow tag \parallel ciphertext$
6: $\mathbf{C} \leftarrow h_4(Enc)$, $n \leftarrow \text{len}(\mathbf{C})$
7: $Init(\text{SHAKE128}(dk_2))$
8: $b_0 = d_o + \text{SHAKE128}(chunk\_size)$
9: $\mathbf{b} \leftarrow [b_i \mid i \in [0, n)]$
where $b_i \leftarrow b_{i-1} + d_o + \mathsf{SHAKE128}(chunk\_size)$
10: Alice chooses a topic *TOPIC*.
11: $Story \leftarrow \text{EmbedderLLM}(\text{LLM}, TOPIC, \varepsilon, T_0, k_0, \mathbf{C}, \mathbf{b}, l, sec)$
12: Alice sends *Story* to Bob.

The second derived key dk_2 is used as a key material to initialize Extendable-Output Function (XOF) SHAKE128 [14] by calling $Init(\text{SHAKE128}(dk_2))$. If SHAKE128 is called again as

$$\text{SHAKE128}(chunk\_size),$$

its output can be seen as an *chunk_size* bits output from the XOF SHAKE128.

For authenticated decryption and verification, Bob does not require the same LLM model as Alice. In fact, he does not need any LLM model at all.

Algorithm 3 LLM Authenticated Decryption and Verification

Input: A shared secret *password* and a text string *Story*
Output: *plaintext* or *Fail*

1: $dk_1, dk_2 \leftarrow \text{PBKDF2}(password, Salt, count, 64)$
2: $Enc \leftarrow$ empty string
3: $Init(\text{SHAKE128}(dk_2))$
4: $pos \leftarrow d_o + \text{SHAKE128}(chunk\_size)$
5: **while** $pos < \text{len}(Story)$ **do**
6: $\quad Enc \leftarrow Enc + Story[pos]$
7: $\quad pos \leftarrow pos + d_o + \text{SHAKE128}(chunk\_size)$
8: **end while**
9: $tag \leftarrow Enc_0, \ldots, Enc_{31}$
10: $tag \leftarrow h_4^{-1}(tag)$
11: $ciphertext \leftarrow Enc_{32}, \ldots, Enc_{last}$
12: $ciphertext \leftarrow h_4^{-1}(ciphertext)$
13: $plaintext \leftarrow AEAD_{dec}(dk_1, nonce, AD, ciphertext, tag)$

Notice the while loop steps 5 – 8 in the decryption algorithm. While Alice does not send the explicit length of *tag* and *ciphertext*, Bob, from the knowledge of *password* and the length of *Story*, can exactly determine *tag* and *ciphertext*.

By embedding AEAD-encrypted messages in LLM-generated text, it remains indistinguishable from normal output, even against advanced "LLM" or "ML classifier" adversaries trained on data consisting of normal/covert stories. The message M is first encrypted with key dk_1, then EmbedderLLM places the

ciphertext in a fluent story guided by dk_2. Randomized token selection and natural phrasing minimize statistical traces, while PBKDF2-derived keys resist brute-force attempts. Without dk_1 and dk_2, attackers cannot detect or recover hidden data, ensuring resilience against steganalysis.

5.2 Public Key LLM Cryptography

Before we describe how our EmbedderLLM algorithm can be used in Public Key Cryptography, let us first prove one simple (although a negative) result concerning a distinguishability of texts *Story* produced by LLMs and EmbedderLLM whether they contain or do not contain embedded cryptographic ciphertext.

Proposition 1. *Let $Story_1$ be generated by an LLM and $Story_2$ be generated by*

$$EmbedderLLM(LLM, TOPIC, \varepsilon, T_0, k_0, \mathbf{C}, \mathbf{b}, l, sec).$$

If the sequence $\mathbf{b}$ *is known to an adversary, then there is an efficient algorithm that distinguishes $Story_1$ and $Story_2$.*

Proof. It is just a simple character extraction. Let us construct two sets $Y_1 = \{C_i|\ Story_1[b_i] = C_i\}$ and $Y_2 = \{C_i|\ Story_2[b_i] = C_i\}$. Since $Story_1$ is generated by an LLM without any constraints, with an overwhelming probability

$$Y_1 \not\subseteq L_i,\ \forall i \in [1, \ldots, 4].$$

On the other hand, we get that

$$Y_2 \subseteq L_l,$$

for some $l \in [1, \ldots, 4]$. □

In a public key setup, the positions of characters $\mathbf{b}$ should be known to both key exchange parties. This implies that $\mathbf{b}$ should be publicly known information accessible also to the adversary. While this might seem like a negative result, there are a lot of use cases and cryptographic protocols that separate the phases of initial (covert exchange of information) key exchange and public key DH key exchange for generating session keys. The initial exchange of some information between communicating parties, which is unavailable to the adversary, is a plausible assumption. The plausibility comes from the fact that in modern society, there are numerous possibilities to perform that first phase of the initial covert information exchange, such as via mobile networks or personal physical meetings combined with websites that dynamically publish data or randomness beacons [33]. We list two cases that use this plausible assumption:

1. The popular messenger application Signal [23] uses the "Double Ratchet Algorithm". It needs a root key that is subsequently used to renew and generate short-lived session keys. The session key generation combines Diffie−Hellman key exchange (DH) and a key derivation function (KDF). The root key is assumed to be in possession of both parties and is outside the definition of the Double Ratchet Algorithm. It can be in the form of so-called "pre-keys" or established with so-called "triple Diffie−Hellman key exchange (3-DH)".

2. The anamorphic encryption [31, 3] assumes that "double key" dk is somehow sent covertly from Bob to Alice without the knowledge of the adversary.

In our case, once that initial information is shared between Alice and Bob, it can be used to produce sequences $\mathbf{b}$ unknown to the adversary. Yet, we formulate as a challenging open problem the following:

Research Problem 1. *Let $Story_1$ be generated by an LLM. Is it possible to design an algorithm*

$$EmbedderLLM(LLM, TOPIC, Story_0, T_0, k_0, \mathbf{C}, \mathbf{b}, l, sec)$$

that produces $Story_2$, such that when the sequence $\mathbf{b}$ *is known to an adversary $\mathcal{A}$, then the advantage of the adversary $\mathcal{A}$ of distinguishing $Story_1$ and $Story_2$ is negligible?*

Algorithm 4 ECDHE-LLM

Input: Shared parameters for `Curve25519`. Potentially a shared secret *rootkey*. Shared parameters for operations related to LLMs such as l, (here $l = 4$), T_0, k_0.
Output: A shared session key $SharedKey$

Alice and Bob compute
if $rootkey$ **then**
  $dk_1, dk_2 \leftarrow \text{PBKDF2}(rootkey, Salt, count, 64)$
else
  $dk_1, dk_2 \leftarrow \text{PBKDF2}(\varepsilon, Salt, count, 64)$
end if
assert $\text{len}(dk_1) = \text{len}(dk_2) = 256$ bits
$Init(\text{SHAKE128}(dk_1))$
$b_0 = d_o + \text{SHAKE128}(chunk\_size)$
$\mathbf{b_{Alice}} \leftarrow [b_i \mid i \in [0, 64)$
  where $b_i \leftarrow b_{i-1} + d_o + \mathsf{SHAKE128}(chunk\_size)$
$Init(\text{SHAKE128}(dk_2))$
$b_0 = d_o + \text{SHAKE128}(chunk\_size)$
$\mathbf{b_{Bob}} \leftarrow [b_i \mid i \in [0, 64)$
  where $b_i \leftarrow b_{i-1} + d_o + \mathsf{SHAKE128}(chunk\_size)$
$Chat \leftarrow \varepsilon$

Alice computes
A random integer in the range $a \xleftarrow{\$} [1, 2^{255} - 19)$
$comp_{Alice} \leftarrow \texttt{rand\_low\_bound\_fixed\_len\_comp}(64, 3, 10)$
  assert $comp_{Alice} = [aa_0, aa_1, \ldots, aa_9]$ such that $\sum_{j=0}^{9} aa_j = 64$
$x_A = \texttt{Curve25519}(a, x_g)$
$\mathbf{C}_A \leftarrow h_4(x_A)$
$\mathbf{CPart}_A \leftarrow \texttt{Partition}(\mathbf{C}_A, comp_{Alice})$
$\mathbf{bPart_A} \leftarrow \texttt{Partition}(\mathbf{b_{Alice}}, comp_{Alice})$
$Story_A \leftarrow \varepsilon$
Bob computes
A random integer in the range $b \xleftarrow{\$} [1, 2^{255} - 19)$
$comp_{Bob} \leftarrow \texttt{rand\_low\_bound\_fixed\_len\_comp}(64, 3, 10)$
  assert $comp_{Bob} = [bb_0, bb_1, \ldots, bb_9]$ such that $\sum_{j=0}^{9} bb_j = 64$
$x_B = \texttt{Curve25519}(b, x_g)$
$\mathbf{C}_B \leftarrow h_4(x_B)$
$\mathbf{CPart}_B \leftarrow \texttt{Partition}(\mathbf{C}_B, comp_{Bob})$
$\mathbf{bPart_B} \leftarrow \texttt{Partition}(\mathbf{b_{Bob}}, comp_{Bob})$
$Story_B \leftarrow \varepsilon$

$Chat \leftarrow Chat||AliceInitialMessage$
$Chat \leftarrow Chat||BobInitialResponse$
for $j \in [0, 9]$ **do**
  Alice side
  $OldStory_A \leftarrow Story_A$
  $Story_A \leftarrow \textsc{EmbedderLLM}(\text{LLM}, Chat, Story_A, T_0, k_0, \mathbf{C}_{A_j}, \mathbf{bA}_j, l, sec)$
  $AliceInput_j \leftarrow Story_A \setminus OldStory_A$
  $Chat \leftarrow Chat||AliceInput_j$
  Bob side
  $OldStory_B \leftarrow Story_B$
  $Story_B \leftarrow \textsc{EmbedderLLM}(\text{LLM}, Chat, Story_B, T_0, k_0, \mathbf{C}_{B_j}, \mathbf{bB}_j, l, sec)$
  $BobInput_j \leftarrow Story_B \setminus OldStory_B$
  $Chat \leftarrow Chat||BobInput_j$
end for
Alice side
  $Story_B \leftarrow BobInput_0|| \ldots ||BobInput_9$
  $PublicKeyBob \leftarrow \{C_i \mid Story_B[\mathbf{b_{Bob}}[i]] = C_i, i \in [0, 64)\}$
  $PublicKeyBob \leftarrow h_4^{-1}(PublicKeyBob)$
  $SharedKeyAlice \leftarrow \texttt{Curve25519}(a, PublicKeyBob)$
Bob side
  $Story_A \leftarrow AliceInput_0|| \ldots ||AliceInput_9$
  $PublicKeyAlice \leftarrow \{C_i \mid Story_A[\mathbf{b_{Alice}}[i]] = C_i, i \in [0, 64)\}$
  $PublicKeyAlice \leftarrow h_4^{-1}(PublicKeyAlice)$
  $SharedKeyBob \leftarrow \texttt{Curve25519}(b, PublicKeyAlice)$
$SharedKey = SharedKeyAlice = SharedKeyBob$
$rootkey \leftarrow SharedKey$
Return $SharedKey$

Let us now describe an Elliptic Curve Diffie−Hellman key exchange with Ephemeral keys (ECDHE)

within our framework. We will use one popular and standardized group of elliptic points, Curve25519. It is defined with the equation

$$y^2 = x^3 + 486662x^2 + x,$$

over the finite field $\mathbb{F}_{2^{255}-19}$. As the generator point $G = (x_g, y_g)$, we take the standard value $G = (9, y_g)$, i.e., $x_g = 9$ where the value of y_g is not necessary to know since it can be computed from the value of x_g. When we write

$$x_A = \texttt{Curve25519}(a, x_g),$$

we mean the x coordinate of the multiplication of G by an integer a that is in the range $[1, 2^{255} - 19)$,

$$(x_A, y_A) = a \cdot G.$$

We can interpret x_A as a 256-bit string (to be used with mapping h_1) or as a sequence of 128 elements in the range [0..3] (appropriate for the use with the mapping h_2) or as e sequence of 64 hexadecimal values (in the range $[0..F]$) (appropriate for the use with the mapping h_4).

When we write

$$comp \leftarrow \texttt{rand\_low\_bound\_fixed\_len\_comp}(n, low, m),$$

it means that $comp$ is a list of m integers greater or equal to low that is a composition of n, i.e., $\texttt{sum}(comp) = n$. In the Algorithm 4, we use arbitrary parameters $low = 3$ and $m = 10$. That means the instruction

$$comp_{Alice} \leftarrow \texttt{rand\_low\_bound\_fixed\_len\_comp}(64, 3, 10),$$

randomly generates a list

$$comp_{Alice} = [aa_0, aa_1, \ldots, aa_9]$$

such that $\sum_{j=0}^{9} aa_j = 64$. Similarly, we produce the list

$$comp_{Bob} = [bb_0, bb_1, \ldots, bb_9]$$

such that $\sum_{j=0}^{9} bb_j = 64$.

The instructions

$$\mathbf{CPart}_A \leftarrow \texttt{Partition}(\mathbf{C}_A, comp_{Alice}),$$
$$\mathbf{bPart_A} \leftarrow \texttt{Partition}(\mathbf{b_{Alice}}, comp_{Alice}),$$

partition $\mathbf{C}_A$ and $\mathbf{b_{Alice}}$ into sublists where $\mathbf{CPart}_A = [\mathbf{C}_{A_0}, \ldots, \mathbf{C}_{A_9}]$ and $\mathbf{bPart_A} = [\mathbf{bA}_0, \ldots, \mathbf{bA}_9]$ according to the composition $comp_{Alice}$ i.e., $len(\mathbf{C}_{A_j}) = len(\mathbf{bA}_j) = aa_j$. Similar applies for

$$\mathbf{CPart}_B \leftarrow \texttt{Partition}(\mathbf{C}_B, comp_{Bob}),$$
$$\mathbf{bPart_B} \leftarrow \texttt{Partition}(\mathbf{b_{Bob}}, comp_{Bob}),$$

where $\mathbf{CPart}_B = [\mathbf{C}_{B_0}, \mathbf{C}_{B_1}, \ldots, \mathbf{C}_{B_9}]$, $\mathbf{bPart_B} = [\mathbf{bB}_0, \ldots, \mathbf{bB}_9]$ and $len(\mathbf{C}_{B_j}) = len(\mathbf{bB}_j) = bb_j$.

By asserting that $Story_A \equiv OldStory_A || AliceInput_j$ with the instruction

$$AliceInput_j \leftarrow Story_A \setminus OldStory_A,$$

we extract the new text $AliceInput_j$ that was appended to $OldStory_A$.

A brief discussion about the steps in Algorithm 4 follows.

Alice and Bob share parameters for `Curve25519`. If they had a previously shared secret key (denoted as *rootkey*), they might use it to calculate the embedding positions. They also share parameters for LLM operations, such as l (a default value is $l = 4$), T_0, and k_0. The first 12 steps calculate the positions where the characters related to the ephemeral public keys of Alice and Bob will be embedded. Then, in steps 13–19, Alice computes her public key x_A, which is mapped to the embeddable characters $\mathbf{C}_A$. Due to the specifics of `Curve25519`, when h_4 is used, the size of the public keys is 64 characters. The goal is to transfer (later) LLM-generated messages that embed those 64 characters in an interactive chat style. For that, in Step 14, Alice needs to generate a random composition $comp_{Alice}$ of the number 64 into m summands (here, we take

$m = 10$ summands, but it can be some other suitable values). Each summand is lower bounded by some value (here, we take $low = 3$). According to $comp_{Alice}$ Alice in Steps 17 and 18 computes the partition lists $\mathbf{CPart}_A$ and $\mathbf{bPart_A}$. Alice resets her story as an empty string in Step 19. Bob conducts similar computing actions in steps 20 – 26.

The ephemeral key exchange is initiated by Alice in Step 27 and is followed by the initial response from Bob. Those two initial messages do not contain embedded public key characters.

Then, in an interactive chat session of $m = 10$ messages (back and forward), Alice and Bob gradually embed their public keys, calling the EMBEDDERLLM() function, trying to continue the stories $Story_A$ and $Story_B$ correspondingly. The smaller incremental parts of the components of the stories $Story_A$ and $Story_B$ are depicted as $AliceInput_j$ and $BobInput_j$ for $j \in [0..9]$. In the final stage (steps 39–42), Alice extracts Bob's chat inputs, extracts his public key, and computes the shared key. Bob performs similar actions in steps 43- 46. The shared keys should be equal, and the value of the *rootkey* is set to that shared key for some future session.

We can emphasize several properties of Algorithm 4, which portray the features of our overall framework:

5.2.1 LLM agnostic

Alice and Bob can use their own locally trained LLMs. The specific local LLMs are invoked in steps 31 and 35. There is no need for those LLMs to be the same.

5.2.2 Pre- or post-quantum agnostic

While we used pre-quantum elliptic curve public key cryptography as an example, we can easily use some post-quantum key exchange schemes. The expected price will be longer chat sessions for transmitting the ephemeral public keys of the communicating parties.

5.2.3 Big number of parameter customizations for further mitigation the distinguishing of an encrypted communication

Since the goal is for the chat sessions to conduct undetectable encrypted communication for any totalitarian regime that monitors people's chats, our default parameters $l = 4$, $m = 10$, $low = 3$, as well as the choice of the public key scheme can be additionally chosen by the participants. The choice of the sets of embeddable characters L_1, L_2, L_3, L_4 can also be a parameter that can have some variations. Another variant with enriched parameters can be a variant where in between those m Alice-Bob chat-pairs-sentences that hold the embedded characters, there are chat-pairs that are just there for confusion and do not hold any embedded characters.

6 Conclusion

In this work, we introduced a novel cryptographic framework that enables secure and covert communication using LLMs. Our approach facilitates encrypted messaging-whether through Public Key or Symmetric Key encryption - while maintaining indistinguishability from natural human-like text in public chat environments. A key advantage of our framework is its LLM agnosticism, allowing participants to use different local models independently. Additionally, it remains resilient against both pre- and post-quantum adversaries, ensuring long-term security. By seamlessly integrating encryption with human-like text generation, our method provides an alternative for secure communication in scenarios where conventional encryption mechanisms are easily detected or restricted.

Implementing our framework remains as a future work. Furthermore, there are several promising avenues for future research. One immediate direction is to solve the Research Problem 1 stated in this work. Another potential direction is the integration of error-correcting codes [22] into our framework. If the bits in the secret positions $\mathbf{b}$ are flipped, the received secret message may deviate from the intended one. Exploring efficient methods to incorporate error correction could enhance the robustness of our approach. Another important direction is optimizing the framework for real-time covert messaging, improving both efficiency

and scalability. Investigating broader applications of our framework in privacy-preserving systems could further expand its impact and usability.

7 Ethical Declaration and Consideration

The work presented, including theoretical formulations and cryptographic constructions, is original, except for minor language refinements in the introduction and related work sections, where ChatGPT was used. These refinements do not influence the technical contributions or the core results of the paper. This research does not include human subjects, personal data, or ethical risks.

References

[1] Statistical Distributions of English Text — web.archive.org. `https://web.archive.org/web/20200205183157/http://www.data-compression.com:80/english.html`, 2020. [Accessed 16-03-2025].

[2] Minhao Bai, Jinshuai Yang, Kaiyi Pang, Yongfeng Huang, and Yue Gao. Shifting-merging: Secure, high-capacity and efficient steganography via large language models. *arXiv preprint arXiv:2501.00786*, 2025.

[3] Fabio Banfi, Konstantin Gegier, Martin Hirt, Ueli Maurer, and Guilherme Rito. Anamorphic encryption, revisited. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–32, Cham, 2024. Springer, Springer Nature Switzerland.

[4] Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024.

[5] J. Black. *Authenticated Encryption*, pages 145–156. Springer Nature Switzerland, Cham, 2025.

[6] Davide Carnemolla, Dario Catalano, Emanuele Giunta, and Francesco Migliaro. Anamorphic resistant encryption: the good, the bad and the ugly. Cryptology ePrint Archive, Paper 2025/233, 2025.

[7] Dario Catalano, Emanuele Giunta, and Francesco Migliaro. Anamorphic encryption: New constructions and homomorphic realizations. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 33–62, Cham, 2024. Springer, Springer Nature Switzerland.

[8] Dario Catalano, Emanuele Giunta, and Francesco Migliaro. Limits of black-box anamorphic encryption. In *Annual International Cryptology Conference*, pages 352–383, Cham, 2024. Springer, Springer Nature Switzerland.

[9] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference, 2024.

[10] Miranda Christ, Sam Gunn, and Or Zamir. Undetectable watermarks for language models. In *The Thirty Seventh Annual Conference on Learning Theory*, pages 1125–1139, Proceedings of Machine Learning Research, 2024. PMLR.

[11] Aloni Cohen, Alexander Hoover, and Gabe Schoenbach. Watermarking language models for many adaptive users. Cryptology ePrint Archive, Paper 2024/759, 2024.

[12] Javier Conde, Gonzalo Martinez, Pedro Reviriego, Zhen Gao, Shanshan Liu, and Fabrizio Lombardi. Can ChatGPT Learn to Count Letters? . *Computer*, 58(03):96–99, March 2025.

[13] Abdulrahman Diaa, Toluwani Aremu, and Nils Lukas. Optimizing adaptive attacks against content watermarks for language models. *arXiv preprint arXiv:2410.02440*, 2024.

[14] Morris J Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions. 2015.

[15] Hugging Face Enterprise. Models - Hugging Face. `https://huggingface.co/models`, March 2025. [Accessed 18-03-2025].

[16] Bill Goodwin. Warning over privacy of encrypted messages as russia targets signal messenger. *Computer Weekly*, 2025.

[17] Yu-Shin Huang, Peter Just, Krishna Narayanan, and Chao Tian. Od-stega: Llm-based near-imperceptible steganography via optimized distributions. *ArXiv*, abs/2410.04328, 2024.

[18] Yu-Shin Huang, Chao Tian, Krishna Narayanan, and Lizhong Zheng. Relatively-secure llm-based steganography via constrained markov decision processes. *arXiv preprint arXiv:2502.01827*, 2025.

[19] Zhengan Huang, Gongxian Zeng, Xin Mu, Yu Wang, and Yue Yu. Multi-designated detector watermarking for language models. *arXiv preprint arXiv:2409.17518*, 2024.

[20] Yuqing Liang, Jiancheng Xiao, Wensheng Gan, and Philip S Yu. Watermarking techniques for large language models: A survey. *arXiv preprint arXiv:2409.00089*, 2024.

[21] Aiwei Liu, Leyi Pan, Yijian Lu, Jingjing Li, Xuming Hu, Xi Zhang, Lijie Wen, Irwin King, Hui Xiong, and Philip Yu. A survey of text watermarking in the era of large language models. *ACM Computing Surveys*, 57(2):1–36, 2024.

[22] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error-correcting codes*, volume 16. Elsevier, 1977.

[23] Moxie Marlinspike. Signal on the outside, Signal on the inside. `https://signal.org/blog/signal-inside-and-out/`, March 2016. [Accessed 19-03-2025].

[24] Dan Milmo. Uk demands ability to access apple usersâĂŹ encrypted data. *The Guardian*, 2025.

[25] Le Monde. Casser le chiffrement de whatsapp ou signal, un serpent de mer politique dangereux. *Le Monde*, 2025.

[26] Kathleen Moriarty, Burt Kaliski, and Andreas Rusch. PKCS #5: Password-Based Cryptography Specification Version 2.1. RFC 8018, January 2017.

[27] OpenAI. Chatgpt: Optimizing language models for dialogue, 2022. Accessed: 2025-02-14.

[28] OpenAI. What are tokens and how to count them?, 2023. Accessed: 2025-03-16.

[29] Qi Pang, Shengyuan Hu, Wenting Zheng, and Virginia Smith. Attacking llm watermarks by exploiting their strengths. In *ICLR 2024 Workshop on Secure and Trustworthy Large Language Models*, 2024.

[30] Qi Pang, Shengyuan Hu, Wenting Zheng, and Virginia Smith. No free lunch in llm watermarking: Trade-offs in watermarking design choices. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[31] Giuseppe Persiano, Duong Hieu Phan, and Moti Yung. Anamorphic encryption: private communication against a dictator. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 34–63, Cham, 2022. Springer, Springer Nature Switzerland.

[32] Vasile-Daniel PĂČvĂČloaia and Sabina-Cristiana Necula. Artificial intelligence as a disruptive technologyâĂŤa systematic literature review. *Electronics*, 12(5), 2023.

[33] Mayank Raikwar and Danilo Gligoroski. SoK: Decentralized randomness beacon protocols. In *Australasian Conference on Information Security and Privacy*, pages 420–446, Cham, 2022. Springer, Springer Nature Switzerland.

[34] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasaman Bahri, Maha Elbayad, Baptiste RoziÃÍre, Dmytro Okhonko, Armand Joulin, Guillaume Lample, and Edouard Grave. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[35] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[36] Yaofei Wang, Gang Pei, Kejiang Chen, Jinyang Ding, Chao Pan, Weilong Pang, Donghui Hu, and Weiming Zhang. Sparsamp: Efficient provably secure steganography based on sparse sampling, 2025.

[37] Yihao Wang, Ruiqi Song, Lingxiao Li, Ru Zhang, and Jianyi Liu. Dairstega: Dynamically allocated interval-based generative linguistic steganography with roulette wheel, 2024.

[38] Wikipedia. Letter frequency - Wikipedia — en.wikipedia.org. `https://en.wikipedia.org/wiki/Letter_frequency`, 2025. [Accessed 12-02-2025].

[39] xAI. Grok 3, 2025. Artificial intelligence model developed by xAI.

[40] Or Zamir. Excuse me, sir? your language model is leaking (information). *arXiv preprint arXiv:2401.10360*, 2024.

[41] Ruisi Zhang, Shehzeen Samarah Hussain, Paarth Neekhara, and Farinaz Koushanfar. {REMARK-LLM}: A robust and efficient watermarking framework for generative large language models. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 1813–1830, Philadelphia, PA, 2024. USENIX Association.

[42] Ruisi Zhang, Neusha Javidnia, Nojan Sheybani, and Farinaz Koushanfar. Robust and secure code watermarking for large language models via ml/crypto codesign. *arXiv preprint arXiv:2502.02068*, 2025.

Appendices

A Algorithm 4 in a protocol format

Algorithm 4 ECDHE-LLM

Input: Shared parameters for Curve25519. Potentially a shared secret $rootkey$. Shared parameters for operations related to LLMs such as l, (here $l = 4$), T_0, k_0.
Output: A shared session key $SharedKey$

Alice	Chat	Bob
if $rootkey$ **then**	$Chat \leftarrow \varepsilon$	**if** $rootkey$ **then**
$dk_1, dk_2 \leftarrow \text{PBKDF2}(rootkey, Salt, count, 64)$		$dk_1, dk_2 \leftarrow \text{PBKDF2}(rootkey, Salt, count, 64)$
else		**else**
$dk_1, dk_2 \leftarrow \text{PBKDF2}(\varepsilon, Salt, count, 64)$		$dk_1, dk_2 \leftarrow \text{PBKDF2}(\varepsilon, Salt, count, 64)$
end if		**end if**
assert $\text{len}(dk_1) = \text{len}(dk_2) = 256$ bits		**assert** $\text{len}(dk_1) = \text{len}(dk_2) = 256$ bits
$Init(\text{SHAKE128}(dk_1))$		$Init(\text{SHAKE128}(dk_1))$
$b_0 = d_o + \text{SHAKE128}(chunk\_size)$		$b_0 = d_o + \text{SHAKE128}(chunk\_size)$
$\mathbf{b}_{\text{Alice}} \leftarrow [b_i \mid i \in [0, 64)$ where $b_i \leftarrow b_{i-1} + d_o + \text{SHAKE128}(chunk\_size)$		$\mathbf{b}_{\text{Alice}} \leftarrow [b_i \mid i \in [0, 64)$ where $b_i \leftarrow b_{i-1} + d_o + \text{SHAKE128}(chunk\_size)$
$Init(\text{SHAKE128}(dk_2))$		$Init(\text{SHAKE128}(dk_2))$
$b_0 = d_o + \text{SHAKE128}(chunk\_size)$		$b_0 = d_o + \text{SHAKE128}(chunk\_size)$
$\mathbf{b}_{\text{Bob}} \leftarrow [b_i \mid i \in [0, 64)$ where $b_i \leftarrow b_{i-1} + d_o + \text{SHAKE128}(chunk\_size)$		$\mathbf{b}_{\text{Bob}} \leftarrow [b_i \mid i \in [0, 64)$ where $b_i \leftarrow b_{i-1} + d_o + \text{SHAKE128}(chunk\_size)$
A random integer in the range $a \xleftarrow{\$} [1, 2^{255} - 19)$		A random integer in the range $b \xleftarrow{\$} [1, 2^{255} - 19)$
$comp_{Alice} \leftarrow \text{rand\_low\_bound\_fixed\_len\_comp}(64, 3, 10)$		$comp_{Bob} \leftarrow \text{rand\_low\_bound\_fixed\_len\_comp}(64, 3, 10)$
assert $comp_{Alice} = [aa_0, aa_1, \ldots, aa_9]$ such that $\sum_{j=0}^{9} aa_j = 64$		**assert** $comp_{Bob} = [bb_0, bb_1, \ldots, bb_9]$ such that $\sum_{j=0}^{9} bb_j = 64$
$x_A = \text{Curve25519}(a, x_g)$		$x_B = \text{Curve25519}(b, x_g)$
$\mathbf{C}_A \leftarrow h_4(x_A)$		$\mathbf{C}_B \leftarrow h_4(x_B)$
$\mathbf{CPart}_A \leftarrow \text{Partition}(\mathbf{C}_A, comp_{Alice})$		$\mathbf{CPart}_B \leftarrow \text{Partition}(\mathbf{C}_B, comp_{Bob})$
$\mathbf{bPart}_A \leftarrow \text{Partition}(\mathbf{b}_{\text{Alice}}, comp_{Alice})$		$\mathbf{bPart}_B \leftarrow \text{Partition}(\mathbf{b}_{\text{Bob}}, comp_{Bob})$
$Story_A \leftarrow \varepsilon$		$Story_B \leftarrow \varepsilon$
$AliceInitialMessage \longrightarrow$	$AliceInitialMessage$	
	$BobInitialResponse \longleftarrow$	$BobInitialResponse$
for $j \in [0, 9]$ **do**		**for** $j \in [0, 9]$ **do**
$OldStory_A \leftarrow Story_A$		
$Story_A \leftarrow \text{EmbedderLLM}(\text{LLM}, Chat, Story_A, T_0, k_0, \mathbf{C}_{A_j}, \mathbf{bA}_j, l, sec)$		
$AliceInput_j \leftarrow Story_A \setminus OldStory_A$		
$AliceInput_0 \longrightarrow$	$AliceInput_0$	$OldStory_B \leftarrow Story_B$
		$Story_B \leftarrow \text{EmbedderLLM}(\text{LLM}, Chat, Story_B, T_0, k_0, \mathbf{C}_{B_j}, \mathbf{bB}_j, l, sec)$
		$BobInput_j \leftarrow Story_B \setminus OldStory_B$
	$BobInput_0 \longleftarrow$	$BobInput_0$
⋮	⋮	⋮
$OldStory_A \leftarrow Story_A$		
$Story_A \leftarrow \text{EmbedderLLM}(\text{LLM}, Chat, Story_A, T_0, k_0, \mathbf{C}_{A_j}, \mathbf{bA}_j, l, sec)$		
$AliceInput_j \leftarrow Story_A \setminus OldStory_A$		
$AliceInput_9 \longrightarrow$	$AliceInput_9$	
		$OldStory_B \leftarrow Story_B$
		$Story_B \leftarrow \text{EmbedderLLM}(\text{LLM}, Chat, Story_B, T_0, k_0, \mathbf{C}_{B_j}, \mathbf{bB}_j, l, sec)$
		$BobInput_j \leftarrow Story_B \setminus OldStory_B$
	$BobInput_9 \longleftarrow$	$BobInput_9$
end for		**end for**
$Story_B \leftarrow BobInput_0 \| \ldots \| BobInput_9$		$Story_A \leftarrow AliceInput_0 \| \ldots \| AliceInput_9$
$PublicKeyBob \leftarrow \{C_i \mid Story_B[\mathbf{b}_{\text{Bob}}[i]] = C_i, i \in [0, 64)\}$		$PublicKeyAlice \leftarrow \{C_i \mid Story_A[\mathbf{b}_{\text{Alice}}[i]] = C_i, i \in [0, 64)\}$
$PublicKeyBob \leftarrow h_4^{-1}(PublicKeyBob)$		$PublicKeyAlice \leftarrow h_4^{-1}(PublicKeyAlice)$
$SharedKeyAlice \leftarrow \text{Curve25519}(a, PublicKeyBob)$		$SharedKeyBob \leftarrow \text{Curve25519}(b, PublicKeyAlice)$
$rootkey \leftarrow SharedKeyAlice$		$rootkey \leftarrow SharedKeyAlice$

B A toy-example of a ECDHE-LLM as a proof of concept

As a proof of concept for ECDHE-LLM let us use the small didactical elliptical curve `secp112r1`. Its parameters are the following: $p = (2^{128} - 3)/76439$, and the elliptic curve E defined over $\mathbb{F}_p$ is the curve

$$\begin{aligned} y^2 =& x^3 + a_4 x + a_6, \\ a_4 =& 4451685225093714772084598273548424, \\ a_6 =& 2061118396808653202902996166388514 \\ G =& (188281465057972534892223778713752, 3419875491033170827167861896082688). \end{aligned}$$

Let us assume that Alice and Bob had agreed (on a slightly changed and permuted L_4 from the set defined in Section 4) set of embeddable characters:

$$L_4 = \{\texttt{' '}, \texttt{E}, \texttt{T}, \texttt{A}, \texttt{O}, \texttt{I}, \texttt{N}, \texttt{S}, \texttt{H}, \texttt{R}, \texttt{D}, \texttt{L}, \texttt{C}, \texttt{U}, \texttt{M}, \texttt{W}\}.$$

Using L_4, for this toy example, the public keys of Alice and Bob will consist of just 28 characters. Let other shared values be $m = 3$ and $low = 7$.

Shortening further the text of this example, let us assume that both Alice and Bob have calculated in steps 1–18 the following sequences:

$$\mathbf{b_{Alice}} = [20, 34, 50, 66, 86, 100, 122, 140, 153, 167, 177, 197, 220, 234, 249, 269, 291, 308, 325, 343, 359, 381, 395, 416, 427, 437, 455, 478],$$

$$\mathbf{b_{Bob}} = [15, 40, 51, 64, 88, 99, 113, 134, 152, 176, 198, 222, 245, 267, 291, 302, 312, 332, 355, 368, 392, 417, 440, 454, 468, 483, 508, 531].$$

Let Alice randomly generated the following:

$$\begin{aligned} a = 128702414016962948846040005401504 9, \\ comp_{Alice} = [7, 10, 11]. \end{aligned}$$

Then, the values of x_A, $\mathbf{CPart}_A$ and $\mathbf{bPart_A}$ will be

$$\begin{aligned} x_A =& \texttt{0x373ac0abcfdf4cf30b9a0cfbe5d6}, \\ \mathbf{CPart}_A =& [[\texttt{A}, \texttt{S}, \texttt{A}, \texttt{D}, \texttt{C}, \texttt{' '}, \texttt{D}], [\texttt{L}, \texttt{C}, \texttt{W}, \texttt{U}, \texttt{W}, \texttt{O}, \texttt{C}, \texttt{W}, \texttt{A}, \texttt{' '}], [\texttt{L}, \texttt{R}, \texttt{D}, \texttt{' '}, \texttt{C}, \texttt{W}, \texttt{L}, \texttt{M}, \texttt{I}, \texttt{U}, \texttt{N}]], \\ \mathbf{bPart_A} =& [[20, 34, 50, 66, 86, 100, 122], [140, 153, 167, 177, 197, 220, 234, 249, 269, 291], [308, 325, 343, 359, 381, 395, 416, 427, 437, 455, 478]]. \end{aligned}$$

Similarly, let us suppose that Bob has generated the following values:

$$\begin{aligned} b =& 144798607543130032972532994815456 0, \\ comp_{Bob} =& [9, 9, 10], \\ x_B =& \texttt{0x7dc525c3afa5307918fb6026d144}, \\ \mathbf{CPart}_B =& [[\texttt{U}, \texttt{S}, \texttt{D}, \texttt{M}, \texttt{U}, \texttt{S}, \texttt{T}, \texttt{O}, \texttt{W}], [\texttt{M}, \texttt{O}, \texttt{R}, \texttt{L}, \texttt{' '}, \texttt{E}, \texttt{R}, \texttt{A}, \texttt{E}], [\texttt{S}, \texttt{A}, \texttt{U}, \texttt{O}, \texttt{' '}, \texttt{D}, \texttt{H}, \texttt{U}, \texttt{' '}, \texttt{T}]], \\ \mathbf{bPart_B} =& [[15, 40, 51, 64, 88, 99, 113, 134, 152], [176, 198, 222, 245, 267, 291, 302, 312, 332], [355, 368, 392, 417, 440, 454, 468, 483, 508, 531]]. \end{aligned}$$

The following chat conversation resembles what would be a generated chat by Algorithm 4.

```
Alice: Hey Bob, what's up? How was your day?
Bob  : Hey Alice! It was good, thanks. Just got out of a meeting at work. You know how it is. How about you?

Alice: I am OK too :-). I want to find a suitable criteria for measuring daily productivity. Can you inform our employees of the duties they have?
Bob  : Definitely. I suppose you want to discuss the records of server maintenance in the core unit. I am sure you will trace all necessary boot ups and shutdowns :-)

Alice: All that is accurate Bob. I want also unit members of the Windows department to collect every access record, twined with customer analytics of purchases and expenditure.
Bob  : I guess that approximately half of the records will be without accreditations, but properly dealing with twined information can still be useful. Are you unsatisfied with the age of the cloud servers?

Alice: Lately, our servers (both Ubuntu and Windows) crash more than usual. Agency DY that we work for can help us locate Autum cases being problematic. Audit records on server nodes that crashed will be analyzed.
Bob  : Bos, your plan arrives on time. Four Ubuntu servers are already on the upgrade schedule for early Monday, two for Thursday, and Thursday next week the rest of them. It seems you tackle the problems successfully.

Alice: Thank you, Bob. It seems we have a plan.
Bob  : Yeah. See you on Monday.
```

Figure 1: The chat between Alice and Bob with highlighted embedded characters and positions

C Adversarial Models and Future Directions

We formalize adversarial models to analyze the security of our framework. These models extend classical cryptographic notions to account for the unique properties of LLM-generated text and the threat of steganalysis.

C.1 Indistinguishability for Covert Communication (IND-CC)

Definition 1 (IND-CC Game)**.** *Let $\mathcal{A}$ be a probabilistic polynomial-time (PPT) adversary and $\mathcal{C}$ a challenger. The IND-CC game proceeds as follows:*

1. ***Setup:***
   - *$\mathcal{C}$ generates $dk_1, dk_2 \leftarrow PBKDF2(password, Salt, count, 64)$.*
   - *$\mathcal{C}$ initializes $SHAKE128(dk_2)$ to sample $\mathbf{b}$ using (5).*
2. ***Query Phase:***
   - *$\mathcal{A}$ adaptively submits messages $\{m_i\}$. For each m_i, $\mathcal{C}$ computes $Enc_i \leftarrow AEAD_{enc}(dk_1, nonce, AD, m_i)$.*
   - *$\mathcal{C}$ returns $Story_i \leftarrow$ EmbedderLLM$(LLM, TOPIC, \varepsilon, T_0, k_0, h_4(Enc_i), \mathbf{b}_i, l, sec)$.*
3. ***Challenge:***
   - *$\mathcal{A}$ submits m^*. $\mathcal{C}$ flips a bit $b \in \{0, 1\}$.*
   - *If $b = 1$, $\mathcal{C}$ returns $Story^* \leftarrow$ EmbedderLLM(m^*).*
   - *If $b = 0$, $\mathcal{C}$ returns $Story^* \leftarrow LLM(TOPIC)$ (no embedding).*
4. ***Guess:*** *$\mathcal{A}$ outputs $b' \in \{0, 1\}$.*

*The scheme achieves **IND-CC security** if $\mathcal{A}$'s advantage*

$$\mathsf{Adv}_{\mathcal{A}}^{IND\text{-}CC} = \left| \Pr[b' = b] - \frac{1}{2} \right|$$

is negligible under Theorem 1.

Relation to Theorem 1: Theorem 1 bounds the probability of tokens deviating from the LLM's optimal parameter ranges ($T \in [0.7, 0.9]$, $k \in [40, 60]$). IND-CC security holds if SHAKE128 is pseudorandom and EmbedderLLM's outputs are indistinguishable from natural text (Table 3).

C.2 Steganographic Secrecy Against Adaptive Adversaries (SS-ADV)

Definition 2 (SS-ADV Game)**.** *Let $\mathcal{A}$ interact with an oracle $\mathcal{O}$:*

- *$\mathcal{O}.\mathsf{Embed}(m)$: Returns $Story \leftarrow$ EmbedderLLM(m).*
- *$\mathcal{O}.\mathsf{Gen}(TOPIC)$: Returns $Story \leftarrow LLM(TOPIC)$.*

*$\mathcal{A}$ adaptively queries $\mathcal{O}$ and guesses its mode. The scheme achieves **SS-ADV security** if $\mathcal{A}$'s distinguishing advantage*

$$\mathsf{Adv}_{\mathcal{A}}^{SS\text{-}ADV} = \left| \Pr[\mathcal{A}^{\mathcal{O}.\mathsf{Embed}} = 1] - \Pr[\mathcal{A}^{\mathcal{O}.\mathsf{Gen}} = 1] \right|$$

is negligible.

Alignment with Framework: SS-ADV security relies on the uniformity of $\mathbf{C}$ characters in L_4 (Table 1) and pseudorandom $\mathbf{b}$ offsets. Adversaries cannot statistically distinguish $h_4(Enc)$-embedded text from natural sequences (see digram frequencies in Table 2).

C.3 Public Key Security (ECDHE-LLM)

Definition 3 (PK-DH Security)**.** *Let $\mathcal{A}$ observe a transcript Chat containing public keys $\mathbf{C}_A = h_4(x_A)$ and $\mathbf{C}_B = h_4(x_B)$ embedded via EmbedderLLM (Algorithm 4). $\mathcal{A}$'s goal is to compute $\texttt{Curve25519}(a, x_B)$. The scheme is **PK-DH-secure** if $\mathcal{A}$'s success probability is negligible, assuming:*

- *The hardness of ECDH on Curve25519.*
- *The IND-CC security of $\mathbf{C}_A, \mathbf{C}_B$ embeddings.*

Relation to Proposition 1: If $\mathbf{b}$ is leaked (Proposition 1), PK-DH security requires $h_4(x_A)$ to be pseudorandom in L_4. Otherwise, security follows directly from ECDH.

C.4 Token Statistical Analysis (TOK-STAT)

Definition 4 (TOK-STAT Distinguishability)**.** *Let $\mathcal{A}$ know the LLM's token distribution $\mathcal{D}_{LLM}$ and receive Story generated via EmbedderLLM. $\mathcal{A}$ wins if it detects statistical deviations (e.g., via KL divergence). The scheme is **TOK-STAT-secure** if:*

$$\mathsf{SD}\left(\mathcal{D}_{Embedder}, \mathcal{D}_{LLM}\right) \leq 2^{-sec},$$

where SD is the statistical distance and sec is the security parameter (Table 3).

Link to Theorem 1: Theorem 1 ensures TOK-STAT security by bounding the probability of sampling tokens outside $T \in [0.7, 0.9]$ and $k \in [40, 60]$.

C.5 Future Research Directions

- **Known-Position Resistance:** Address Open Problem 1 by dynamically masking $\mathbf{b}$ or injecting dummy embeddings.
- **Post-Quantum LLM Cryptography:** Integrate lattice-based KEMs (e.g., Kyber) and formalize indistinguishability under quantum queries.
- **Efficiency Optimizations:** Accelerate EmbedderLLM's token search (Steps 19–41, Algorithm 1) using GPU parallelism.
- **Multimodal Covert Channels:** Extend the framework to image/video LLMs (e.g., DALL–E, Sora) for cross-modal embeddings.