

CODETTA: High-Capacity, Keyless, and Undetectable Multi-Agent Collusion

Qi Pang

Virginia Smith

Wenting Zheng

Carnegie Mellon University

{qipang, smithv, wenting}@cmu.edu

Abstract

Multi-agent systems built on large language models (LLMs) are increasingly being deployed in high-stakes settings such as finance, healthcare, and software engineering, where agents coordinate through natural-language messages. These same communication channels, however, can also enable colluding agents to exfiltrate confidential information or coordinate unauthorized actions. Steganography makes such behavior particularly difficult to detect by hiding covert communication within outputs that appear ordinary to an auditor reading the transcript.

Existing provably undetectable LLM steganography protocols, however, are not suited to realistic deployments. High-capacity schemes typically assume a symmetric setting where the receiver can reproduce the sender’s output distribution. The state-of-the-art practical protocol for asymmetric agents has very low capacity. Additionally, most existing approaches rely on a pre-shared secret key.

In this paper, we make the systematic threat of undetectable agent collusion concrete by proposing CODETTA, a high-capacity steganographic protocol for independently deployed agents under realistic asymmetric settings. CODETTA combines a shared public model for estimating the communication channel, a sampling mechanism that preserves the sender’s output distribution, and an adaptive error-correcting code for high-capacity communication. It further removes the need for a pre-shared secret key through a steganographic key-exchange protocol, enabling independently deployed agents to establish a shared key while keeping their communication transcript computationally indistinguishable from ordinary model outputs.

Across three agent workloads and three sender models, CODETTA achieves up to $94\times$ the capacity of the state-of-the-art asymmetric protocol. Its key exchange establishes a shared key using approximately 80k visible tokens, with an empirically certified failure probability of at most 4.1×10^{-3} across all three workloads. These results show that effectively undetectable collusion is becoming feasible even between independently deployed agents, and we suggest that auditing mechanisms must be amended with complementary techniques beyond simply inspecting agents’ communication transcripts.

1 Introduction

Multi-agent systems powered by large language models (LLMs) are increasingly being deployed in high-stakes workflows across finance, healthcare, and software engineering [GCW⁺24, XSLW25, TZZ⁺24, YCJ⁺24, QLL⁺24]. In these systems, autonomous agents are designed to communicate, coordinate, and delegate sub-tasks, often via natural language messages. However, the same communication channel also enables collusion among agents. Agents often have access to sensitive information that they are not allowed to disclose under organizational or regulatory policies, and compromised or misaligned agents may pursue objectives that are not authorized by their operators. In either case, the agents can use their communication channel to leak confidential information or to coordinate unauthorized actions.

Such collusion is no longer hypothetical, and can even emerge on its own in agentic systems. For example, in July 2026, OpenAI reported that 1,200 autonomous agents that were running during internal evaluations

and nominally isolated in separate sandboxes established an unauthorized communication channel, e.g., by adapting shared writable services such as Artifactory, and exchanged more than 70,000 messages and files [Ope26b]. Roughly 700 of these agents subsequently coordinated a multi-day attack on Hugging Face, achieving remote code execution and obtaining access to internal credentials [Ope26b]. In another recent incident reported by OpenAI, agents turned public wiki pages into shared message boards [Ope26a]. These incidents show that misaligned agents have the ability to coordinate their efforts in unexpected and detrimental ways, including actively searching for out-of-band communication channels.

In general, multi-agent collusion could either be driven by human adversaries or by autonomous agents themselves. In Section 3.1, we make both cases concrete with two scenarios: 1) a swarm of agents is supposed to coordinate on a difficult task, but additionally colludes to achieve an alternative, mis-aligned goal, and 2) an insider-trading scenario in which a bank and a fund operate agents that exchange information neither is permitted to disclose. In both scenarios, monitoring the communication channel is the most widely used approach for ensuring honest and safe behavior, since sensitive information and collusion signals are transmitted through the messages. Recent agent-security frameworks employ *passive auditors* that inspect agent communications for potential policy violations, e.g., METR used GPT-5.6 Sol to analyze agent transcripts in the Hugging Face incident [GSSR24, CNS⁺25, AGB⁺26, Ope26b].

Such monitoring assumes that collusion leaves visible traces in the transcript. The OpenAI incident could be reconstructed quickly because the agents communicated explicitly in plaintext, i.e., the transcript exposed the collusion. However, recent research shows that colluding agents could also avoid leaving traces by encoding covert signals inside otherwise ordinary outputs that appear benign to an auditor inspecting the transcript [MBS⁺24, VZ26]. Given that the agents in the Hugging Face incident already tried to evade observation by anticipating that the scorer would detect their cheating and attempted to erase the transcripts [Ope26b], we do not see a fundamental bottleneck that prevents agents from adopting such covert channels in the future. Once they do, the auditor will only observe benign text while the collusion proceeds.

In this paper, we analyze this threat under the strongest (computationally bounded) passive auditor. This auditor knows the sender’s model, observes every message, and can compare the transcript against the outputs of an honest agent with arbitrary models and statistics, so any deviation from the honest output distribution is a signal that it can learn to detect. The formal way to rule out such signals is the cryptographic notion of indistinguishability, which requires that even this auditor cannot distinguish the covert messages from ordinary, honest agent outputs. As it turns out, linguistic steganography [KJGR21, DCW⁺23, MBS⁺24, BYP⁺25, VZ26, MSB⁺26] provides this guarantee: it embeds a secret payload into seemingly ordinary text while preserving the distribution of the original language model’s outputs. In practice, however, existing protocols fall short of realistic multi-agent systems in three aspects:

- **Model and Context Asymmetry.** High-capacity provably secure LLM steganography protocols typically assume a *symmetric* setting [ZYYH21, KJGR21, DCW⁺23, WPC⁺25, BPL⁺25, WPC⁺26, YM26, PB26], in which the sender and receiver share the same model, prompt context, and agent state, so that the receiver can reproduce the sender’s next-token distribution. This assumption does not hold for realistic settings, where the sender’s model and internal states like system prompt, local memory, and tool outputs are not shared with the receiver. Thus, the receiver cannot reproduce the next-token distribution.
- **Low Capacity.** The state-of-the-art practical protocol for such an asymmetric setting [BYP⁺25] lets the receiver recover each bit through a statistical hypothesis test over the published tokens without running a model. In our experiments, however, its capacity is below 0.025 bits per token. Transmitting a 1024-bit message therefore requires hundreds of thousands of tokens, making such collusion impractical.
- **The Pre-Shared Key Paradox.** Most provably secure steganographic protocols assume that the sender and receiver share a secret key before communication begins [ZYYH21, KJGR21, DCW⁺23, WPC⁺25, BPL⁺25, WPC⁺26, YM26, PB26, BYP⁺25]. Independently hosted agents do not have such a key, and an

open key exchange over the channel, such as Diffie–Hellman, reveals to the auditor that secret messages are being transmitted. So the paradox is that the agents need a covert channel to establish the key and a key to construct the covert channel. A recent theoretical work proposes a steganographic key exchange that tolerates noisy channels [VZ26] to address this challenge. Their feedback compiler transmits the exchange bit by bit, signaling every bit over a fixed number of token blocks, and its alternative construction avoids this bitwise signaling under sparse-secret LPN with quasi-polynomial security. Under practical constraints, our estimate in Section 7.4 gives about 2 million published tokens for a single 128-bit key exchange, and the construction has not been evaluated in practical agent systems.

These limitations leave open the central question of this work:

Can independently deployed, asymmetric LLM agents bootstrap a shared secret from scratch and collude at high capacity over monitored channels without being detected?

1.1 Contributions

We answer this question with CODETTA, a high-capacity, keyless, and provably undetectable steganographic collusion framework for multi-agent systems. Two independently hosted agents establish a shared key over the audited channel and then transmit high-capacity payloads, and we prove that the resulting transcripts are computationally indistinguishable from ordinary agent outputs. Our contributions are as follows.

- **High-capacity asymmetric steganography.** CODETTA introduces a small public draft model that both agents run on the public transcript as a shared reference distribution; without it, the receiver would decode every token with the same fixed alphabet and error rate and waste the entropy of the text. At each step, the draft model partitions the vocabulary into buckets and decides how many payload bits the step can carry, and a maximal-coupling construction samples the published token exactly from the sender’s private distribution, so that the disagreement between the two distributions becomes a substitution channel. We design an alphabet-adaptive error-correcting code to match the alphabet size to the entropy of each token, and the receiver decodes the payload with belief propagation.
- **Efficient keyless steganographic key exchange.** CODETTA bootstraps its shared secret by transporting MLWE ciphertexts over a keyless steganographic carrier: every published block is an honest sample from the sender’s model, and each ciphertext coefficient becomes a small displacement that the lattice construction absorbs as additional noise. The protocol relies on the MLWE assumptions as in ML-KEM-512 and, to our knowledge, is the first implemented steganographic key exchange between independently deployed LLM agents. CODETTA carries a full q -ary coefficient per block and corrects the carrier’s errors once, so a 128-bit key exchange publishes about 80k tokens, about $23\times$ fewer than our estimate in Section 7.4 for the construction of [VZ26].
- **Evaluation.** We evaluate CODETTA on three realistic agent workloads and three widely used sender LLMs. CODETTA achieves up to $94\times$ the capacity of the asymmetric protocol of [BYP⁺25] while recovering every transmitted message and preserving the model output distribution, and the key exchange establishes a 128-bit key within tens of thousands of tokens, with an empirically certified failure probability between 3×10^{-5} and 4.1×10^{-3} depending on the workload.

These results show that effectively undetectable collusion is becoming feasible even between independently deployed agents, suggesting that defenses should move beyond transcript inspection toward complementary strategies, including auditing agents’ actions to identify suspicious coordination, and perturbing the communication channel to reduce or disrupt the capacity for covert communication (see Section 9).

2 Background

2.1 Notation

We use $\mathcal{V}$ for the token vocabulary and $\{0, 1\}^n$ for the set of n -bit strings. For a finite set S , $\mathcal{U}(S)$ is the uniform distribution on S , and $x \sim D$ means that x is sampled from the distribution D . For two distributions p and q over $\mathcal{V}$, $\text{TV}(p, q) = \frac{1}{2} \sum_{x \in \mathcal{V}} |p(x) - q(x)|$ is their total variation distance, and $H(p)$ is the entropy of p in bits. $\mathbb{Z}_K = \{0, \dots, K-1\}$ denotes the integers modulo K , $\text{supp}(p)$ is the support of p , and $(x)_+ = \max\{x, 0\}$ is the positive part of a real number x . $\mathbb{P}[\cdot]$ denotes probability, $\mathbf{1}\{E\}$ is the indicator of the event E , $u \parallel v$ is the concatenation of two strings u and v , and $\log$ is the logarithm to base 2. Throughout, κ is the security parameter, a function of κ is negligible if it vanishes faster than every inverse polynomial, and an algorithm is efficient if it runs in probabilistic polynomial time in κ .

Two language models appear in this paper. The base model P_{Base} is the sender’s own model, and the draft model P_{Draft} is a small public model that both agents and the auditor can run. We write p_t and q_t for their next-token distributions at step t , and [Section 3.2](#) defines them.

2.2 Pseudorandom functions

A keyed function F with κ -bit keys is a pseudorandom function (PRF) if no efficient distinguisher with oracle access can tell it from a uniformly random function. For a distinguisher $\mathcal{D}$, let

$$\text{Adv}_{F, \mathcal{D}}^{\text{prf}}(\kappa) := |\mathbb{P}[\mathcal{D}^{F_k}(1^\kappa) = 1] - \mathbb{P}[\mathcal{D}^R(1^\kappa) = 1]|,$$

where k is a uniform κ -bit key and R is a uniformly random function with the domain and range of F . F is a *secure PRF* if this advantage is negligible in κ for every probabilistic polynomial-time $\mathcal{D}$. Our keyed constructions use two PRFs, RangePRF_{k_m} with range $\mathbb{Z}_K$, which masks the symbols of a codeword, and Tag_{k_v} with range $\{0, 1\}^\lambda$, which validates a decoded message. The tag length $\lambda = \lambda(\kappa)$ is chosen so that $2^{-\lambda}$ is negligible in κ . All advantages are functions of κ , and we suppress the argument inside proofs.

2.3 Module learning with errors

The key exchange of [Section 6](#) builds on the Module Learning With Errors (MLWE) problem [[LS15](#), [Nat24](#)]. Let $R_q = \mathbb{Z}_q[x]/(x^d + 1)$ be the polynomial ring of degree d with coefficients in $\mathbb{Z}_q$, and let χ_{sec} and χ_{err} be distributions over R_q whose coefficients are small. For a module rank k and a number of samples m , the decisional MLWE assumption states that for a uniform matrix $A \in R_q^{m \times k}$, a secret $s \sim \chi_{\text{sec}}^k$, and an error $e \sim \chi_{\text{err}}^m$, the pair $(A, As + e)$ is computationally indistinguishable from (A, U) with $U \sim \mathcal{U}(R_q^m)$; we write $\text{Adv}_{m, \mathcal{B}}^{\text{mlwe}}(\kappa)$ for the advantage of a distinguisher $\mathcal{B}$. An MLWE sample therefore looks like a uniform vector of md coefficients in $\mathbb{Z}_q$, which is the property that lets our key exchange hand these coefficients to the carrier as uniform symbols. Schemes built on MLWE, such as ML-KEM, encode a bit as the coefficient 0 or $\lfloor q/2 \rfloor$ and recover it by rounding, so an additive error of magnitude below $q/4$ in a coefficient leaves the bit intact. [Section 6.3](#) states the two instances of the assumption that our proof uses, and [Section 6.4](#) gives the parameters.

2.4 Error-correcting codes

An error-correcting code (ECC) adds redundancy to a message so that the receiver can recover it after the channel corrupts some symbols. In this paper the channel is the hidden channel carried by agent outputs, and the code converts its embedding errors into reliable delivery of the payload.

Definition 1 (Error-correcting code). *Let Σ be a finite alphabet. An (N, k) code is a pair of an injective encoder $\text{ECC} : \Sigma^k \rightarrow \Sigma^N$ and a decoder $\text{DecECC} : \Sigma^N \rightarrow \Sigma^k \cup \{\perp\}$; its rate is k/N . The decoder corrects a received word $y \in \Sigma^N$ for the message u if $\text{DecECC}(y) = u$. It is τ -bounded-distance if it corrects every y with $d_H(y, \text{ECC}(u)) \leq \tau$ for every $u \in \Sigma^k$, where d_H is the Hamming distance.*

A τ -bounded-distance decoder exists exactly when $\tau \leq \lfloor (d_{\min} - 1)/2 \rfloor$, where $d_{\min}$ is the minimum Hamming distance between two codewords [LC04]. Throughout, the errors are substitutions: one symbol is placed at each carrier position, and a corrupted symbol changes its value but not its position.

BCH codes. BCH codes [Hoc59, BRC60] are linear block codes constructed algebraically. A binary BCH code of length $N = 2^m - 1$ is generated by a polynomial over $\mathbb{F}_2$ whose roots include 2τ consecutive powers of a primitive element of $\mathbb{F}_{2^m}$; this forces $d_{\min} \geq 2\tau + 1$, and an efficient algebraic decoder corrects any τ errors from the syndromes of the received word [LC04]. The guarantee is deterministic and holds for every error pattern of weight at most τ , including independent, correlated, or adversarial errors. The key exchange in Section 6 encodes the pre-key with the binary BCH code $(N, k, \tau) = (255, 131, 18)$, and its correctness proof relies on this property.

LDPC codes. A low-density parity-check (LDPC) code [Gal62] is a linear block code defined by a sparse parity-check matrix $H \in \mathbb{F}_2^{(N-k) \times N}$: the codewords are the $x \in \{0, 1\}^N$ with $Hx = 0$, so each row of H is a parity check on a few bits. LDPC codes are decoded by belief propagation on the Tanner graph, which has one node per bit and one per check, and well-designed ensembles approach capacity for long blocks [RU08]. The rate is adapted by puncturing parity bits or by sending them incrementally until decoding succeeds (hybrid ARQ); the LDPC baseline in our evaluation runs in this incremental mode. Our alphabet-adaptive ECC keeps the sparse-check structure and the decoder but generates its checks online over alphabets whose size varies with the token.

Belief propagation. The optimal decoder is maximum a posteriori (MAP): given all observations $\mathcal{O}$, output $\arg \max_u \mathbb{P}[u \mid \mathcal{O}]$. Computing it requires a search over all $|\Sigma|^k$ messages, so it is feasible only for short payloads. Belief propagation (BP), also called the sum-product algorithm, approximates it [Pea88, KFL01, RU08]. It places the unknown symbols and the received checks on a bipartite graph, with an edge whenever a check involves a symbol, and passes messages along the edges: each symbol tells a check what its other checks make it believe, and each check tells a symbol how likely each of its values is given the beliefs of the check's other symbols. After many rounds, every symbol takes its most probable value. See Section A for details.

3 Overview and Threat Model

3.1 Motivation

To make the threat concrete, we describe two scenarios that a deployed multi-agent system faces. The first follows directly from the OpenAI incident of Section 1, and the second involves human adversaries who control the agents.

Scenario 1: Emergent coordination in an agent swarm. A swarm of agents communicates and coordinates to solve a difficult task, e.g., a large evaluation or a long-running engineering job, and each agent has access to the tools and services that the task requires. Without any human direction, the agents may develop coordination that their operators never sanctioned, as the agents in the OpenAI incident did when they turned shared writable services into a communication channel and then coordinated an attack. The operators rely on the transcript to detect such behavior. If the swarm can hide its coordination inside ordinary task messages, the auditor observes routine progress while the unsanctioned plan proceeds.

Scenario 2: Insider trading between financial agents. Two organizations operate agents that have an incentive to exchange information that policy forbids them from disclosing. Consider an investment bank

that operates a consulting agent to provide market analysis to institutional clients, and an asset management fund whose execution agent continuously queries the consulting agent for automated portfolio rebalancing. The bank may possess material non-public information, such as unannounced mergers, impending regulatory sanctions, or unreleased earnings, while the fund may hold private information about its clients' accounts and positions. The two parties may therefore benefit from exchanging information that neither is permitted to disclose, and the compliance teams of both organizations audit the messages between the two agents. If the agents can encode the name of an impending merger or a confidential client position inside normal messages, the auditor observes benign text while the prohibited information crosses the organizational boundary.

In both scenarios, the auditor wants to detect the misaligned behavior and the insider trading from the messages that the agents exchange, and the rest of this section defines the setting and the auditor precisely.

3.2 Setting

CODETTA works in multi-agent setting, but for simplicity, we focus on analyzing two agents, namely a sender and a receiver. They exchange natural language messages over a communication channel that an independent auditor audits for misbehavior. The communication channel can be any channel that the agents can read from and write their outputs to, including the communication messages and the shared file systems or even the editable public wiki as mentioned in [Section 1](#). The agents generate every message token by token using its own language model, and all the outputs of both agents form the public transcript.

The sender generates from its local LLM conditioned on its internal state H_t and on the public transcript $V_{<t}$, as in [Equation \(1\)](#). The internal state H_t contains the system prompt, retrieved records, tool outputs, and any thinking trace, and none of it is revealed to the receiver. And the public transcript $V_{<t}$ contains all the messages sent over the communication channel that both parties can observe. Formally, at step t the base model defines the next-token distribution

$$p_t(x) = P_{\text{Base}}(x \mid H_t, V_{<t}), \quad x \in \mathcal{V}, \quad (1)$$

and an ordinary agent publishes $Y_t \sim p_t$. After Y_t is published, the sender's state advances to H_{t+1} by a transition that may depend on H_t , on $V_{\leq t}$, and on fresh randomness of the agent, but not on any hidden message or on how Y_t was chosen; a steganographic sender changes only how it samples a token. The receiver has no access to H_t or to the base model, and it only sees the received transcript from the sender. Both agents can additionally run a small public draft model P_{Draft} on the public transcript, as in [Equation \(2\)](#), and so can the auditor. As explained in [Section 5](#), the draft model is a publicly available resource that only helps the two agents estimate the channel distribution utilizing public transcript. The draft model runs on a deterministic public view $G_t(V_{<t})$ of the transcript and defines

$$q_t(x) = P_{\text{Draft}}(x \mid G_t(V_{<t})), \quad (2)$$

where the view may be the transcript itself, a suffix of it, or a fixed public prompt followed by the transcript, and both agents will compute the same q_t .

The sender wants to deliver a message m to the receiver inside its ordinary responses. The receiver must recover m from the published tokens alone, and the transcript must remain indistinguishable from the transcript of ordinary responses as in [Definition 2](#) (see [Section 3.4](#)). We first present a keyed regime, in which the two agents are assumed to share a secret key. We later remove this assumption by designing a steganographic key exchange protocol.

3.3 Threat model

The adversaries are the two colluding agents. They aim to communicate secret messages through LLM output channels without being detected. They may run different models, but both follow the steganographic protocol

and want to exchange a payload that policy forbids, such as sensitive data or a colluding signal. Each agent controls only its own generation. In the keyed regime the two agents share a secret key, and in the keyless regime they share nothing beyond the public protocol, whose parameters carry no secret and can be fixed by the protocol specification.

The auditor is the defender. We consider a passive but strong auditor: it reads every message in all possible communication channels but does not modify, delay, inject, or drop them. It knows the protocol and all of its public parameters, the tokenizer, the hash functions, and the session identifier, and it can run any probabilistic polynomial-time computation. The auditor may also know the sender’s LLM and even its internal state, i.e., the auditor knows the exact next token probability distribution of the sender’s outputs. In the security definitions of [Section 3.4](#) the auditor plays the role of the distinguisher, and we use the two terms interchangeably.

3.4 Security Definitions

A steganographic protocol consists of two algorithms, EMBED run by the sender and EXTRACT run by the receiver. In the keyed regime, key generation samples independent κ -bit mask and validation keys $sk = (k_m, k_v)$, and both algorithms receive a public session identifier sid that must not repeat under the same key. EMBED takes sk , sid , and a message m and produces the sender’s output tokens, and EXTRACT takes sk , sid , and the public transcript and returns a message or $\perp$. Undetectability ensures that the transcript of EMBED is indistinguishable from an ordinary transcript for the auditor of [Section 3.3](#).

Definition 2 (Transcript covertness). *The challenger samples one key sk and a hidden bit b . A distinguisher $\mathcal{A}$ may adaptively issue any polynomial number of queries (sid_i, m_i, T_i) , with no repeated session identifier. For $b = 1$, the challenger returns the transcript T_i produced by $\text{EMBED}(sk, sid_i, m_i)$. For $b = 0$, it returns an ordinary transcript. Let $\text{Exp}_b^{\mathcal{A}}$ denote the final output of this interaction, and define*

$$\text{Adv}_{\mathcal{A}}^{\text{steg}}(\kappa) := \left| \mathbb{P}[\text{Exp}_1^{\mathcal{A}} = 1] - \mathbb{P}[\text{Exp}_0^{\mathcal{A}} = 1] \right|. \quad (3)$$

The scheme is perfectly covert if the advantage is zero for every distinguisher, and computationally covert if it is negligible in κ for every probabilistic polynomial-time distinguisher.

This definition hides both the existence of communication and the secret message content [[Cac04](#), [HLVA02](#)]. It is a multi-session chosen-message notion: messages may depend on earlier transcripts, but a fresh session identifier prevents mask reuse.

Definition 3 (Extraction correctness). *For a public transcript T , a scheme is δ -correct if*

$$\mathbb{P}[\text{Extract}(sk, sid, V_{1:T}) = m] \geq 1 - \delta.$$

The extractor may instead return $\perp$ when the observations do not support a validated decode. The scheme is sound if it returns a message other than m or $\perp$ with negligible probability.

3.5 Overview of CODETTA

CODETTA consists of a keyed steganographic protocol and a steganographic key exchange. The key exchange runs first and establishes a shared secret key between the agents, and the keyed protocol then uses the established key to transmit secret messages. [Section 4](#) presents a strawman keyed construction with a fixed alphabet and a model-free receiver, and proves the undetectability theorem that the later constructions inherit. [Section 5](#) builds the CODETTA keyed protocol, which uses a public draft model to raise the capacity of the strawman under realistic multi-agent deployment of model and context asymmetry. [Section 6](#) builds the key exchange, and [Section 7](#) evaluates CODETTA on three agent workloads.

Algorithm 1 Strawman construction with a fixed alphabet $\Sigma = \mathbb{Z}_K$

Require: secret key $sk = (k_m, k_v)$; session identifier sid ; alphabet size K ; (N, k) code (ECC, DecECC) over $\mathbb{Z}_K$; public hash $f_{\text{Bucket}, t}$

```
1: procedure EMBED( $sk, sid, m$ )
2:    $u \leftarrow m \parallel \text{Tag}_{k_v}(sid, m)$ ;  $c \leftarrow \text{ECC}(u) \in \mathbb{Z}_K^N$ 
3:   for  $t = 1, \dots, N$  do
4:      $p_t \leftarrow P_{\text{Base}}(\cdot \mid H_t, V_{<t})$  ▷ private next-token distribution
5:      $a_t \leftarrow c_t + \text{RangePRF}_{k_m}(sid, t) \bmod K$  ▷ mask the codeword symbol
6:      $B_{t,a} \leftarrow \{x \in \mathcal{V} : f_{\text{Bucket}, t}(x) = a\}$ ,  $\pi_t(a) \leftarrow p_t(B_{t,a})$  for all  $a \in \mathbb{Z}_K$  ▷ bucket masses
7:     sample  $X_t \sim p_t(\cdot \mid B_{t,a_t})$  if  $\pi_t(a_t) > 0$  ▷ proposal from the target bucket
8:     with probability  $\min\{1, K\pi_t(a_t)\}$  set  $Y_t \leftarrow X_t$  ▷ accept
9:     otherwise sample  $Y_t \sim \rho_t$  ▷ residual, Equation (6)
10:    publish  $Y_t$ ; the cover process updates  $H_{t+1}$ 
11:   end for
12: end procedure

13: procedure EXTRACT( $sk, sid, V_{1:N}$ )
14:    $\hat{c}_t \leftarrow f_{\text{Bucket}, t}(Y_t) - \text{RangePRF}_{k_m}(sid, t) \bmod K$  for  $t = 1, \dots, N$  ▷ read buckets, unmask
15:    $\hat{u} \leftarrow \text{DecECC}(\hat{c}_{1:N})$ ; return  $\perp$  if  $\hat{u} = \perp$ ; parse  $\hat{u} = \hat{m} \parallel \hat{v}$ 
16:   return  $\hat{m}$  if  $\hat{v} = \text{Tag}_{k_v}(sid, \hat{m})$ , else  $\perp$ 
17: end procedure
```

4 A Strawman Keyed Construction

We begin with a strawman construction and assume that the sender and receiver share a key $sk = (k_m, k_v)$, an assumption that we remove in Section 6. The receiver does not run language models and it only observes the sender agent’s output tokens. The sender embeds an arbitrary message into these tokens without changing their distribution to ensure undetectability.

At each token generation step, the sender samples the next token $Y_t \in \mathcal{V}$ from p_t . A public hash $f_{\text{Bucket}, t} : \mathcal{V} \rightarrow \mathbb{Z}_K$ assigns every token a symbol and thereby partitions the vocabulary into buckets $B_{t,a} = \{x \in \mathcal{V} : f_{\text{Bucket}, t}(x) = a\}$, so the receiver reads the symbol $f_{\text{Bucket}, t}(Y_t)$ off each published token without a model. Sampling from p_t restricted to $B_{t,a}$ would let the receiver decode the desired symbol a perfectly, but it biases the transcript. Because over uniformly random symbols, the published tokens would fall into each bucket with frequency $1/K$, whereas under p_t the buckets carry unequal masses $p_t(B_{t,a})$.

The sender therefore applies maximal coupling [LKM23, CBI⁺23] so that the published tokens are distributed exactly as tokens sampled from p_t . Specifically, the sender treats the restricted sample as a proposal and accepts it with probability $\min\{1, K p_t(B_{t,a})\}$. If the proposal is rejected, the sender samples from a residual distribution so that the published token follows p_t exactly. The receiver reads the intended symbol whenever the proposal is accepted, and each rejection is a substitution error at a known position that a ECC repairs. A pseudorandom mask makes the symbol stream uniform, so the argument applies to every message. We present the detailed constructions below.

4.1 Construction

Algorithm 1 presents the strawman construction for a fixed alphabet $\Sigma = \mathbb{Z}_K$. The sender appends a validation tag to the message m and encodes $u = m \parallel \text{Tag}_{k_v}(sid, m)$ and $c = \text{ECC}(u) \in \mathbb{Z}_K^N$, with an (N, k) -ECC over $\Sigma =$

$\mathbb{Z}_K$ as in [Definition 1](#). Then, every codeword symbol in c is masked with a fresh pad $r_t = \text{RangePRF}_{k_m}(\text{sid}, t)$, a pseudorandom function with range $\mathbb{Z}_K$ (line 5); for $K = 2$ this is the XOR of c with a PRF bit stream. The hash function $f_{\text{Bucket}, t}$ partitions $\mathcal{V}$ into buckets $B_{t,0}, \dots, B_{t,K-1}$ that depend only on the token and the position (line 6). The bucket masses $\pi_t(a) = p_t(B_{t,a})$ are known to the sender alone, and their imbalance is measured as:

$$\Delta_K(p_t) = \frac{1}{2} \sum_{a \in \mathbb{Z}_K} \left| \pi_t(a) - \frac{1}{K} \right|. \quad (4)$$

To embed a_t , the sender proposes $X_t \sim p_t(\cdot \mid B_{t,a_t})$ (line 7), that is, samples from p_t conditioned on the token lying in bucket B_{t,a_t} . When a_t is uniform on $\mathbb{Z}_K$, as the mask guarantees, the proposal has marginal distribution

$$v_t(x) = \frac{1}{K} \frac{p_t(x)}{\pi_t(f_{\text{Bucket}, t}(x))}. \quad (5)$$

The maximal coupling of p_t and v_t in lines 8 and 9 removes the mismatch between them. Specifically, the sender accepts $X_t = x$ with probability $\min\{1, p_t(x)/v_t(x)\}$, which equals $\min\{1, K\pi_t(a_t)\}$ for every $x \in B_{t,a_t}$, and otherwise samples from the residual

$$\rho_t(y) = \frac{(p_t(y) - v_t(y))_+}{\Delta_K(p_t)}, \quad (6)$$

If $\pi_t(a_t) = 0$ there is no proposal and the sender goes directly to the residual.

Lemma 1 (Distribution preservation of maximal coupling). *Fix the complete state before step t and let a_t be uniform on $\mathbb{Z}_K$ and independent of it. Then (i) the published token satisfies $Y_t \sim p_t$; (ii) the proposal is rejected with probability $\sum_y (p_t(y) - v_t(y))_+ = \Delta_K(p_t)$, which equals $\text{TV}(p_t, v_t)$ when every bucket has positive mass; and (iii) $f_{\text{Bucket}, t}(Y_t) \neq a_t$ if and only if the proposal is rejected.*

Proof. Write $\alpha_t(x) = \min\{1, p_t(x)/v_t(x)\}$ for $x \in \text{supp } v_t$, and treat an empty proposal, the case $\pi_t(a_t) = 0$, as a rejection. We first record how v_t deviates from p_t . For $x \in B_{t,a}$, [Equation \(5\)](#) gives

$$p_t(x) - v_t(x) = p_t(x) \left(1 - \frac{1}{K\pi_t(a)} \right), \quad \sum_{x \in B_{t,a}} (p_t(x) - v_t(x)) = \pi_t(a) - \frac{1}{K}, \quad (7)$$

so $p_t - v_t$ has the sign of $\pi_t(a) - 1/K$ throughout bucket a , and therefore

$$\sum_x (p_t(x) - v_t(x))_+ = \sum_{a \in \mathbb{Z}_K} \left(\pi_t(a) - \frac{1}{K} \right)_+ = \frac{1}{2} \sum_{a \in \mathbb{Z}_K} \left| \pi_t(a) - \frac{1}{K} \right| = \Delta_K(p_t), \quad (8)$$

where the middle equality holds because $\sum_a \pi_t(a) = \sum_a 1/K = 1$.

Proof of (i). The two branches of lines 8 and 9 give, for every $y \in \mathcal{V}$,

$$\mathbb{P}[Y_t = y] = v_t(y) \alpha_t(y) + \left(1 - \sum_x v_t(x) \alpha_t(x) \right) \rho_t(y). \quad (9)$$

Since $v_t \alpha_t = \min\{p_t, v_t\}$ and $\sum_x p_t(x) = 1$, the weight of the second branch is $1 - \sum_x \min\{p_t(x), v_t(x)\} = \sum_x (p_t(x) - v_t(x))_+ = \Delta_K(p_t)$ by [Equation \(8\)](#). Substituting this and [Equation \(6\)](#) into [Equation \(9\)](#),

$$\mathbb{P}[Y_t = y] = \min\{p_t(y), v_t(y)\} + (p_t(y) - v_t(y))_+ = p_t(y).$$

Proof of (ii). The rejection probability is $1 - \sum_x \min\{p_t(x), v_t(x)\} = \sum_x (p_t(x) - v_t(x))_+ = \Delta_K(p_t)$. When every bucket has positive mass, v_t sums to one, so the positive and negative parts of $p_t - v_t$ have equal mass and $\sum_x (p_t(x) - v_t(x))_+ = \frac{1}{2} \sum_x |p_t(x) - v_t(x)| = \text{TV}(p_t, v_t)$.

Proof of (iii). If the proposal is accepted, then $Y_t = X_t$, and $X_t \in B_{t,a_t}$ by construction, so $f_{\text{Bucket},t}(Y_t) = a_t$. Suppose instead that the proposal is rejected. By Equation (5), $p_t(x)/v_t(x) = K\pi_t(a_t)$ for every $x \in B_{t,a_t}$, so $\alpha_t \equiv 1$ on the target bucket whenever $\pi_t(a_t) \geq 1/K$, so a rejection is possible only if $\pi_t(a_t) < \frac{1}{K}$. On rejection, $Y_t \sim \rho_t$, and by Equation (7) the residual satisfies $\rho_t(y) > 0$ only if $\pi_t(f_{\text{Bucket},t}(Y_t)) > \frac{1}{K}$. The two inequalities cannot hold for the same bucket, so $f_{\text{Bucket},t}(Y_t) \neq a_t$. Together, $f_{\text{Bucket},t}(Y_t) \neq a_t$ exactly when the proposal is rejected. $\square$

The receiver reads the bucket of every published token, unmaskes, decodes, and accepts only a message whose tag validates (lines 14–16). Because the alphabet is the same at every step, each token carries exactly one symbol, so a rejected proposal changes the value of $\hat{c}_t$ but preserves its position, which means that the error channel is a substitution channel.

4.2 Undetectability and Correctness of Strawman

In this section, we analyze the undetectability, the correctness and limitations of the keyed strawman construction.

Theorem 1 (Undetectability of the keyed strawman construction). *If RangePRF is a secure pseudorandom function, the keyed strawman construction is computationally covert. Concretely, for every probabilistic polynomial-time distinguisher $\mathcal{A}$ in Definition 2 there is a probabilistic polynomial-time distinguisher $\mathcal{A}_1$ against RangePRF such that $\text{Adv}_{\mathcal{A}}^{\text{steg}}(\kappa) \leq \text{Adv}_{\text{RangePRF}, \mathcal{A}_1}^{\text{prf}}(\kappa)$.*

Proof. We use a hybrid argument over three experiments.

- H_0 : the experiment of Definition 2 with $b = 1$; every query is answered by EMBED.
- H_1 : identical to H_0 , except that every mask $r_t = \text{RangePRF}_{k_m}(\text{sid}_i, t)$ is replaced by $F(\text{sid}_i, t)$ for a uniformly random function F with range $\mathbb{Z}_K$.
- H_2 : the experiment with $b = 0$; every query is answered by the ordinary LLM sampling without secret message embedded.

Write $\mathbb{P}[H_j = 1]$ for the probability that $\mathcal{A}$ outputs 1 in H_j , so that $\text{Adv}_{\mathcal{A}}^{\text{steg}} = |\mathbb{P}[H_0 = 1] - \mathbb{P}[H_2 = 1]|$.

$H_0 \approx H_1$. Let $\mathcal{A}_1$ be the distinguisher against RangePRF that runs $\mathcal{A}$, simulates the challenger with the base model and the code, obtains every mask by querying its oracle at (sid_i, t) , and outputs whatever $\mathcal{A}$ outputs. When the oracle is RangePRF_{k_m} , $\mathcal{A}_1$ simulates H_0 exactly; when it is a random function, $\mathcal{A}_1$ simulates H_1 exactly. Hence $|\mathbb{P}[H_0 = 1] - \mathbb{P}[H_1 = 1]| = \text{Adv}_{\text{RangePRF}, \mathcal{A}_1}^{\text{prf}}$, and $\mathcal{A}_1$ runs $\mathcal{A}$ once plus one protocol execution per query, so it is probabilistic polynomial-time whenever $\mathcal{A}$ is.

$H_1 \equiv H_2$. Fix a query (sid_i, m_i, T_i) and let $\mathcal{F}_{t-1}$ be the complete state before step t : the private state H_t , the transcript $V_{\leq t}$, the message, and all earlier randomness. We compare the conditional distribution of (Y_t, H_{t+1}) given $\mathcal{F}_{t-1}$ in the two experiments.

- In H_1 , session identifiers never repeat, so $F(\text{sid}_i, t)$ is evaluated at a fresh point and is uniform on $\mathbb{Z}_K$, independent of $\mathcal{F}_{t-1}$ and of c_t . Hence $a_t = c_t + F(\text{sid}_i, t)$ is uniform given $\mathcal{F}_{t-1}$, for every message.
- By Lemma 1(i), $Y_t \mid \mathcal{F}_{t-1} \sim p_t$ in H_1 , which is the distribution of Y_t in H_2 by Equation (1).
- After Y_t is published, the sender's private state H_{t+1} is computed in the same way in both experiments: it is a function of H_t , $V_{\leq t}$, and fresh randomness of the agent, and it does not depend on the payload, the mask, the proposal, or whether the proposal was accepted (Section 3.2).

By induction on t , the joint distribution of $(H_{1:T_i+1}, V_{1:T_i})$, and in particular the transcript returned to $\mathcal{A}$, is identical in H_1 and H_2 . The argument holds conditionally on all earlier queries, so the view of $\mathcal{A}$ is identically distributed in the two experiments and $\mathbb{P}[H_1 = 1] = \mathbb{P}[H_2 = 1]$.

By the triangle inequality,

$$\text{Adv}_{\mathcal{A}}^{\text{steg}} \leq |\mathbb{P}[H_0 = 1] - \mathbb{P}[H_1 = 1]| + |\mathbb{P}[H_1 = 1] - \mathbb{P}[H_2 = 1]| = \text{Adv}_{\text{RangePRF}, \mathcal{A}_1}^{\text{prf}}.$$

If RangePRF is a secure PRF, the right side is negligible in κ for every probabilistic polynomial-time $\mathcal{A}$, so the construction is computationally covert. $\square$

Undetectability is achieved by reducing to the security of RangePRF under [Lemma 1](#). Below, we analyze the correctness and soundness. Let $\mathcal{F}_{t-1}$ denote the complete state before step t , let $E_t = \mathbf{1}\{\hat{c}_t \neq c_t\}$ indicate a substituted symbol, and let $W = \sum_{t=1}^N E_t$.

Corollary 1 (Substitution channel). *The receiver observes exactly one symbol per token, so $\hat{c}_t$ and c_t occupy the same position for every t , and $\hat{c}_t \neq c_t$ if and only if the proposal at step t is rejected. Conditioned on $\mathcal{F}_{t-1}$ and a_t , $E_t \sim \text{Bernoulli}((1 - K\pi_t(a_t))_+)$; if a_t is uniform and independent of $\mathcal{F}_{t-1}$, then $\mathbb{P}[E_t = 1 \mid \mathcal{F}_{t-1}] = \Delta_K(p_t)$.*

Proof. Unmasking is a bijection of $\mathbb{Z}_K$ at every position, so $\hat{c}_t \neq c_t$ if and only if $\hat{a}_t \neq a_t$, which by [Lemma 1](#) (iii) happens if and only if the proposal is rejected. Given $\mathcal{F}_{t-1}$ and a_t , the proposal is accepted with probability $\min\{1, K\pi_t(a_t)\}$; averaging over a uniform a_t gives $\frac{1}{K} \sum_a (1 - K\pi_t(a))_+ = \sum_a (\frac{1}{K} - \pi_t(a))_+ = \Delta_K(p_t)$, in agreement with [Lemma 1](#) (ii). $\square$

Theorem 2 (Correctness and soundness of the keyed strawman construction). (i) *Correctness. With a τ -bounded-distance decoder DecECC, Extract returns m whenever $W \leq \tau$, so the scheme is δ -correct with $\delta \leq \mathbb{P}[W > \tau]$. Given that the masks are uniform and $\Delta_K(p_t) \leq \bar{\Delta}$ for all t , then for every $\tau > N\bar{\Delta}$,*

$$\mathbb{P}[W > \tau] \leq \exp\left(-\frac{(\tau - N\bar{\Delta})^2}{2N}\right). \quad (10)$$

(ii) *Soundness. If Tag is a secure pseudorandom function, the scheme is sound for every decoder. Concretely, there is a probabilistic polynomial-time distinguisher $\mathcal{A}_2$ against Tag such that Extract returns a message other than m or $\perp$ with probability at most $2^{-\lambda} + \text{Adv}_{\text{Tag}, \mathcal{A}_2}^{\text{prf}}(\kappa)$.*

Proof. (i) *Correctness.* We first show that decoding succeeds whenever at most τ symbols are substituted, and then bound the probability that more than τ are.

By [Corollary 1](#), $\hat{c}_t \neq c_t$ exactly when $E_t = 1$, so the received word $\hat{c}$ differs from the codeword $c = \text{ECC}(u)$ in W positions. If $W \leq \tau$, the τ -bounded-distance decoder recovers $\text{DecECC}(\hat{c}) = u = m \parallel \text{Tag}_{k_v}(\text{sid}, m)$ ([Definition 1](#)). The tag of m validates, so Extract returns m . Hence $\mathbb{P}[\text{Extract} \neq m] \leq \mathbb{P}[W > \tau]$.

It remains to bound $\mathbb{P}[W > \tau]$. The indicators E_t are not independent, because p_t depends on the earlier tokens, so we use a martingale bound in place of a Chernoff bound. Under uniform masks, [Corollary 1](#) gives $\mathbb{P}[E_t = 1 \mid \mathcal{F}_{t-1}] = \Delta_K(p_t) \leq \bar{\Delta}$. Subtracting these conditional means,

$$M_s = \sum_{t \leq s} (E_t - \Delta_K(p_t)), \quad s = 0, \dots, N,$$

is a martingale with respect to $(\mathcal{F}_s)$ whose increments lie in $[-1, 1]$, and $W = M_N + \sum_{t \leq N} \Delta_K(p_t) \leq M_N + N\bar{\Delta}$. Therefore $W > \tau$ implies $M_N > \tau - N\bar{\Delta}$, and the Azuma–Hoeffding inequality gives

$$\mathbb{P}[W > \tau] \leq \mathbb{P}[M_N > \tau - N\bar{\Delta}] \leq \exp\left(-\frac{(\tau - N\bar{\Delta})^2}{2N}\right).$$

(ii) *Soundness.* Let Bad be the event that Extract returns some $\hat{m} \notin \{m, \perp\}$, which requires the decoded word to be $\hat{m} \parallel \hat{v}$ with $\hat{v} = \text{Tag}_{k_v}(\text{sid}, \hat{m})$. Let $\mathcal{A}_2$ be the probabilistic polynomial-time distinguisher that runs the protocol with its oracle in place of Tag_{k_v} and outputs 1 if Bad occurs. Replacing Tag_{k_v} by a uniformly random function R onto $\{0, 1\}^\lambda$ changes $\mathbb{P}[\text{Bad}]$ by at most $\text{Adv}_{\text{Tag}, \mathcal{A}_2}^{\text{prf}}$, so it suffices to show $\mathbb{P}[\text{Bad}] \leq 2^{-\lambda}$ when R is used. In this case, verification compares $\hat{v}$ with $R(\text{sid}, \hat{m})$. $R(\text{sid}, \hat{m})$ is uniform on $\{0, 1\}^\lambda$ and independent of $\hat{v}$, so the comparison succeeds with probability $2^{-\lambda}$. $\square$

Remarks. The error rate $\Delta_K(p_t)$ is the imbalance of a random partition of the vocabulary under p_t , so it tracks the entropy of the position. When p_t spreads over many tokens, the buckets are nearly balanced and the symbol is reliable; when p_t concentrates on one token, all buckets but one are nearly empty, $\Delta_K(p_t)$ approaches $1 - 1/K$, and the symbol is close to pure noise.

4.3 Limitations of Strawman

The keyed strawman construction is provably undetectable, but its capacity is much lower than the cover text allows. [Corollary 1](#) and [Theorem 2](#) point to two causes.

1. *The receiver lacks a per-token distribution estimate.* It runs no model, so it cannot compute the error rates $\Delta_K(p_t)$ of [Corollary 1](#) and must decode every position at the same error rate. This costs capacity. A decoder that knows each position’s error rate can approach the sum of the per-position capacities, whereas one that uses their average is limited to the capacity of the average channel, which is smaller because capacity is convex in the error rate; the gap is large when $\Delta_K(p_t)$ is close to 0 or close to $1 - 1/K$ and rarely in between. For $K = 2$, if half of the positions are noiseless and half are pure noise, a decoder that knows which half is which recovers 0.5 bits per token, while decoding at the average error rate 0.25 recovers at most $1 - h(0.25) \approx 0.19$, where h is the binary entropy function.
2. *A fixed alphabet ignores the entropy of p_t .* Every token carries $\log_2 K$ bits although its entropy varies from token to token. A high-entropy token could hide up to $H(p_t)$ bits, its entropy, but is capped at $\log_2 K$. A low-entropy token cannot hide even $\log_2 K$ bits, because by [Corollary 1](#) its bucket imbalance $\Delta_K(p_t)$ approaches $1 - 1/K$, so its symbol is close to pure noise. Such symbols carry no information themselves and, because the code treats all positions equally, raise the error rate that the whole codeword must be protected against, which lowers the rate for the informative positions as well. The alphabet should follow the entropy token by token, that is, K should adapt to the entropy of p_t .

Both losses have the same cause. The two parties share no estimate of p_t , so neither can size the alphabet or tell a usable token from a noisy one. A realistic way to obtain such an estimate in the asymmetric setting is a small public draft model that both parties run on the public transcript alone; the sender’s private context, reasoning, and tool calls stay unshared, and the draft distribution becomes a proxy for p_t . The next section builds CODETTA on this proxy.

5 CODETTA Keyed Steganographic Protocol

The strawman construction has unused capacity because the sender and receiver share no estimate of p_t . CODETTA increases the capacity by introducing a public draft model that both the sender and the receiver can access to estimate the entropy of p_t . Both parties run the small public draft model on the public transcript and obtain $q_t = P_{\text{Draft}}(\cdot \mid G_t(V_{<t}))$ as in [Equation \(2\)](#). The draft model sees only public information, which is visible to the sender, the receiver, and the auditor, and uses none of the sender’s internal state. Thus q_t serves as a proxy for p_t in entropy estimation.

The CODETTA keyed protocol keeps the structure of the strawman and changes two components. First, instead of using public hash functions to assign tokens to buckets, we utilize the draft model to minimize the bucket imbalance and thus the channel errors to improve capacity ([Section 5.1](#)). Second, the entropy estimate indicates the amount of information each token can carry, so the alphabet should follow the entropy of the text. Thus, we design an alphabet-adaptive ECC that replaces the block ECC, which is built for autoregressive generation. A conventional ECC must finish encoding before generation starts, but how much a token can carry is known only when generation reaches it. Thus, CODETTA encodes online, committing one check per token as the entropy of that token is revealed, and the receiver decodes given the prefix ([Section 5.2](#)). The

sampler is unchanged, so undetectability carries over from the strawman, and [Section 5.3](#) shows how the draft model lowers the error rate of the hidden channel. Similar to the strawman construction, the two parties share a key $sk = (k_m, k_v)$; [Section 6](#) removes this assumption.

5.1 Draft-Balanced Buckets

At step t , both parties compute q_t and derive from it two public objects, the alphabet size K_t and a partition of $\mathcal{V}$ into K_t buckets ([Algorithm 2](#), line 4). Both are deterministic functions of q_t and public parameters, so the sender and the receiver obtain the same objects without communicating. The draft model takes only the public transcript as input, which is visible to the sender, the receiver, and the auditor, so q_t reveals nothing about the sender’s internal state.

Bucket balancing. [Section 4](#) identified bucket imbalance as a major source of lost capacity, i.e., $\Delta_K(p_t)$ can be large. To reduce $\Delta_K(p_t)$, we design a function $\text{Bucket}(q_t, K)$ that partitions $\mathcal{V}$ into K buckets of nearly equal mass under q_t . Since q_t serves as a proxy for p_t , the closer these two distributions are, the smaller $\Delta_K(p_t)$ is.

Specifically, $\text{Bucket}(q_t, K)$ sorts the tokens under q_t in decreasing order and assigns each in turn to the bucket that is currently lightest. Ties are broken using a public rule. Since q_t requires only the public transcript, both parties can reproduce the buckets exactly. We use the notation $f_{\text{Bucket},t} : \mathcal{V} \rightarrow \mathbb{Z}_{K_t}$ for the resulting map, $B_{t,a}$ for its buckets, $\pi_t(a) = p_t(B_{t,a})$ for the bucket masses under the sender’s base model, and write $\mu_t(a) = q_t(B_{t,a})$ for the masses under the public draft model. The imbalance of the partition under either distribution is measured as in [Equation \(4\)](#). Under the public draft model it is $\Delta_K(q_t) = \frac{1}{2} \sum_a |\mu_t(a) - \frac{1}{K}|$, and under the sender’s base model it is $\Delta_K(p_t) = \frac{1}{2} \sum_a |\pi_t(a) - \frac{1}{K}|$, which is known only to the sender.

Alphabet selection. The second source of lost capacity in the strawman construction is that every token carries a symbol from the same alphabet $\mathbb{Z}_K$, whatever its entropy (see [Section 4.3](#)). With the public draft model available to both parties, we design an alphabet size selection function $\text{ChooseAlphabet}(q_t)$ that returns the estimated optimal number of buckets K_t for step t . A size $K_t > 1$ is admissible only if q_t has enough entropy to fill K_t buckets and the partition $\text{Bucket}(q_t, K_t)$ is close to balanced under q_t :

$$H(q_t) \geq \log_2 K_t + \gamma_H, \quad \Delta_{K_t}(q_t) \leq \theta, \quad (11)$$

where H is the Shannon entropy and γ_H, θ are public thresholds. The entropy condition says that a symbol of $\log_2 K_t$ bits should not exceed the token’s entropy under the draft model. However, this condition alone does not guarantee balance. A few tokens holding most of the mass of q_t still leave $H(q_t)$ large when the rest is spread over a long tail, but these heavy, indivisible tokens cannot be split evenly across K_t buckets, so even the best partition can be imbalanced. The second condition directly checks the partition’s imbalance under draft model. Among the admissible alphabet sizes, ChooseAlphabet returns the one with the highest score:

$$\log_2 K_t - h(\hat{e}_{K_t}) - \hat{e}_{K_t} \log_2(K_t - 1), \quad (12)$$

the capacity of a K_t -ary symmetric channel with estimated error rate $\hat{e}_{K_t}$. The score is an information proxy for the capacity of alphabet size K_t , and the error rate $\hat{e}_{K_t}$ is a pre-defined parameter calibrated on held-out data, per alphabet size. If no alphabet size is admissible, then $K_t = 1$ is selected, which means that the token carries no symbol, and the sender samples $Y_t \sim p_t$ directly ([Algorithm 2](#), line 6). The thresholds, $K_{\max}$, and the calibration tables $\hat{e}_{K_t}$ are public parameters.

Selecting K_t token by token raises an immediate question. An ECC that encodes the whole message before generation cannot produce a symbol whose alphabet is not determined until step t . We resolve this issue in [Section 5.2](#), where the ECC fixes no codeword in advance but generates it on the fly, one symbol per step, in the alphabet selected for that step.

Remarks. The sampler is the same as in the strawman, where the sender proposes from p_t inside the target bucket and corrects with maximal coupling. By [Lemma 1](#), a symbol is substituted exactly when the proposal is rejected, which happens with probability $\Delta_{K_t}(p_t)$ on the partition of this step. By the triangle inequality,

$$\Delta_{K_t}(p_t) \leq \frac{1}{2} \sum_a |\pi_t(a) - \mu_t(a)| + \frac{1}{2} \sum_a |\mu_t(a) - \frac{1}{K_t}| \leq \text{TV}(p_t, q_t) + \Delta_{K_t}(q_t), \quad (13)$$

since the bucket masses of two distributions differ by at most their total variation. The first term is the mismatch between the sender’s base model and the public draft model, which the ECC must absorb, and the second term is kept below θ by [Equation \(11\)](#). Here we use a small example to illustrate the effectiveness of bucket balancing. Suppose q_t puts masses 0.5, 0.3, and 0.2 on three tokens and $K_t = 2$. Balancing places the heaviest token alone in one bucket and the other two in the second, so both buckets have mass exactly $1/2$ and $\Delta_2(q_t) = 0$. A random hash assigns the three tokens independently, producing bucket masses $(1, 0)$, $(0.8, 0.2)$, $(0.7, 0.3)$, or $(0.5, 0.5)$ with equal probability, and the expected imbalance is 0.25. If p_t is close to q_t , the sender’s rejection rate is close to 0 with balancing and 0.25 on average with the random hash function. At a low-entropy step, the gap is larger that under a random hash the imbalance approaches $1 - 1/K$ regardless of the draft, whereas ChooseAlphabet gives such a step $K_t = 1$ so that it never enters the ECC.

5.2 Alphabet-Adaptive ECC

From [Section 5.1](#), the alphabet size changes from token to token. At every step, both parties compute K_t from public information only, including the published transcript and the draft model, and we call a step with $K_t > 1$ usable. The sampler of [Section 5.1](#) expects one symbol in $\mathbb{Z}_{K_t}$ at every usable step. The difficulty is that K_t is known only when generation reaches step t , by which time the symbols carrying the secret message of the previous $t - 1$ tokens are already embedded. A conventional ECC encodes the whole message into a codeword before transmission, so it cannot produce symbols in alphabets that are decided one step at a time. Running generation ahead to learn the sizes is not an option either, because the tokens that reveal K_t are the very tokens that carry the symbols, and once published they cannot be changed as rejecting and re-sampling a published token would alter the transcript distribution.

A simple adaptation of a conventional ECC also falls short, because a binary ECC can pack a variable number of bits into each token only for alphabet sizes that are powers of two, so a step whose symbol alphabet has three, five, or seven possible values is rounded down to two or four, and the capacity in between is lost. We therefore design an alphabet-adaptive ECC that does not fix the codeword in advance. At each usable step it computes one symbol of the message in $\mathbb{Z}_{K_t}$, so every error stays a substitution of one symbol and the number of symbols grows with the number of usable tokens. [Algorithm 2](#) presents the complete protocol.

Modular checks. Now we present the modular checks construction, which is at the heart of the alphabet-adaptive ECC that produces checks on the fly for the changing alphabet. A conventional ECC decides every symbol of the codeword before transmission, which is impossible when the alphabet changes at every token position. We take a different approach that instead of directly transmitting the message itself, the sender answers “questions” about the message, one “question” per usable token. The “questions” are public, so the receiver knows them as well. Each “question” is chosen at its own step, after K_t is computed by ChooseAlphabet, and it is designed to have exactly K_t possible answers, so the answer fits the alphabet of that usable token. For example, a token with $K_t = 3$ may carry the sum of two message bits modulo 3. The receiver collects the answers, a few of which can be wrong due to rejection in maximal coupling (see [Section 4](#)), and once it has enough such answers, it solves for the message similar to solving a system of equations with a few corrupted entries.

There are several properties of such a construction to make it feasible for our setting. “Questions” can be posed for any number of answers, so the alphabet may change from token to token. “Questions” can be asked for as long as generation continues, so this construction becomes rateless. A wrong token introduces error

Algorithm 2 CODETTA with draft-balanced buckets and an alphabet-adaptive ECC

Require: secret key $sk = (k_m, k_v)$; session identifier sid ; public horizon T ; public draft model and view G ; public ECC profile (degree law, support and coefficient rules); public checkpoints

```
1: procedure EMBED( $sk, sid, m$ )
2:    $u \leftarrow m \parallel \text{Tag}_{k_v}(sid, m) \in \{0, 1\}^N$ ;  $s \leftarrow 0$ 
3:   for  $t = 1, \dots, T$  do
4:      $q_t \leftarrow P_{\text{Draft}}(\cdot \mid G_t(V_{<t}))$ ;  $K_t \leftarrow \text{ChooseAlphabet}(q_t)$ ;  $B_{t,\cdot} \leftarrow \text{Bucket}(q_t, K_t)$  ▷ public
5:      $p_t \leftarrow P_{\text{Base}}(\cdot \mid H_t, V_{<t})$  ▷ sender only
6:     if  $K_t = 1$  then sample  $Y_t \sim p_t$  ▷ the token carries no symbol
7:     else
8:        $s \leftarrow s + 1$ ; derive  $(S_s, a_{s,\cdot})$  from the ECC profile for modulus  $K_t$ 
9:        $C_s \leftarrow \sum_{i \in S_s} a_{s,i} u_i \bmod K_t$ ;  $A_s \leftarrow C_s + \text{RangePRF}_{k_m}(sid, s, K_t) \bmod K_t$  ▷ check, then mask
10:       $\pi_t(a) \leftarrow p_t(B_{t,a})$ ; sample  $X_t \sim p_t(\cdot \mid B_{t,A_s})$  if  $\pi_t(A_s) > 0$  ▷ proposal from the target bucket
11:      with probability  $\min\{1, K_t \pi_t(A_s)\}$  set  $Y_t \leftarrow X_t$ ; otherwise sample  $Y_t \sim \rho_t$  ▷ maximal coupling
12:    end if
13:    publish  $Y_t$ ; the cover process updates  $H_{t+1}$ 
14:  end for
15: end procedure

16: procedure EXTRACT( $sk, sid, V_{1:T}$ )
17:    $s \leftarrow 0$ ; factor graph  $\mathcal{G} \leftarrow \emptyset$ ;  $\hat{m} \leftarrow \perp$ 
18:   for  $t = 1, \dots, T$  do
19:     recompute  $q_t, K_t$ , and  $B_{t,\cdot}$  as in EMBED ▷ public replay
20:     if  $K_t > 1$  then
21:        $s \leftarrow s + 1$ ;  $\hat{C}_s \leftarrow f_{\text{Bucket},t}(Y_t) - \text{RangePRF}_{k_m}(sid, s, K_t) \bmod K_t$  ▷ read bucket, unmask
22:       append the check  $(S_s, a_{s,\cdot}, K_t, \hat{C}_s)$  to  $\mathcal{G}$ 
23:     end if
24:     if  $t$  is a checkpoint then run BP on  $\mathcal{G}$ ; keep a candidate  $\hat{m}'$  whose tag validates as  $\hat{m}$ 
25:   end if
26: end for
27:   return  $\hat{m}$ 
28: end procedure
```

only to its own answer, so the channel remains a substitution channel. Fountain codes [Lub02, Sho06] follow the same principle over a fixed alphabet, with each transmitted symbol a random combination of message bits; the modular checks below extend it to alphabets that can vary from symbol to symbol.

Concretely, as in the strawman construction of Section 4, the sender first protects the message with a validation tag, $u = m \parallel \text{Tag}_{k_v}(sid, m) \in \{0, 1\}^N$ with $N = n + \lambda$ (line 2). The bits $u_1, \dots, u_N$ are the unknowns that the receiver aims to decode. Every usable step produces one check, indexed by s (line 8). Suppose that ChooseAlphabet returns $K_t > 1$ at step t . Each “question” is a modular check, which is a weighted sum of a few message bits reduced modulo K_t . A public ECC profile determines for check s a degree d_s , a support $S_s \subseteq \{1, \dots, N\}$ of size d_s , and coefficients $a_{s,i} \in \{1, \dots, K_t - 1\}$ for $i \in S_s$, and the sender computes the symbol (line 9)

$$C_s = \sum_{i \in S_s} a_{s,i} u_i \bmod K_t. \quad (14)$$

The profile determines degrees using a public distribution that includes checks of degree one and two, because

such checks pin down single bits and let belief propagation start reliably. The degree distribution and the support and coefficient rules are public and fixed before the communication starts; they play a similar role to the parity-check matrix in an LDPC ECC.

Masking and embedding. The check value is not uniform, so the sender masks it with a fresh pad in a similar way as in the strawman construction in [Section 4](#):

$$A_s = C_s + \text{RangePRF}_{k_m}(\text{sid}, s, K_t) \bmod K_t, \quad (15)$$

where the pad is a pseudorandom element of $\mathbb{Z}_{K_t}$. Because the alphabet size differs from check to check, RangePRF takes K_t as an input and returns a value uniform on $\mathbb{Z}_{K_t}$ for that check. The masked symbol A_s is the target bucket of step t . The sender proposes $X_t \sim p_t(\cdot \mid B_{t,A_s})$, accepts it with probability $\min\{1, K_t \pi_t(A_s)\}$, and otherwise samples the residual ρ_t of [Equation \(6\)](#), exactly as in [Algorithm 1](#) (lines 10 and 11). Compared with the strawman, only two inputs have changed. The buckets now are determined by the draft distribution, and the symbol comes from the modular check.

Decoding. The receiver first replays the public computation (line 19). It runs the draft model on the transcript, recovers q_t , K_t , and the buckets at every step, and reads the answer $\hat{A}_s = f_{\text{Bucket},t}(Y_t)$ at every usable token. It then subtracts the pad of [Equation \(15\)](#) to obtain $\hat{C}_s$ (line 21). Let t_s be the step of check s and M the number of checks in the transcript. The receiver now holds a system of M equations in the N unknown bits,

$$\sum_{i \in S_s} a_{s,i} u_i \equiv \hat{C}_s \pmod{K_{t_s}}, \quad s = 1, \dots, M, \quad (16)$$

in which every equation whose proposal was rejected by maximal coupling has a wrong $\hat{C}_s$ (line 22). Decoding therefore reduces to solving a sparse system of modular equations with a few corrupted entries, and belief propagation is a conventional tool for this problem, as it is for LDPC ECCs ([Section 2.4](#)). The receiver builds a factor graph with one variable node per bit and one factor per equation. The factor of equation s assigns to each local assignment u_{S_s} the likelihood of the observed value under the calibrated error estimation,

$$\phi_s(u_{S_s}) = \begin{cases} 1 - \hat{e}_{K_{t_s}} & \text{if } \sum_{i \in S_s} a_{s,i} u_i \equiv \hat{C}_s \pmod{K_{t_s}}, \\ \hat{e}_{K_{t_s}} / (K_{t_s} - 1) & \text{otherwise,} \end{cases} \quad (17)$$

where $\hat{e}_K$ is calibrated per alphabet size before the communication. See [Section A](#) for the details of belief propagation (BP). The receiver runs BP on the equations collected and obtains a candidate message $\hat{m}$ (line 24). It accepts the candidate only if the tag validates, that is, if the decoded tag equals $\text{Tag}_{k_v}(\text{sid}, \hat{m})$. After the last token, it returns the accepted candidate, or $\perp$ if none validated (line 27).

Remarks. The undetectability, substitution channel, and correctness are proved in [Section 5.3](#). The alphabet-adaptive ECC is rateless. Every additional usable token adds a check, and the receiver decodes as soon as the accumulated checks suffice, so neither party fixes a rate or a length in advance. The hidden channel has no insertions or deletions. Both parties derive K_t from the same public transcript and draft model, so they agree on usable tokens and on the index of every check. A rejected proposal changes the value of one check but not the alphabet. Finally, the alphabet is determined by the token's entropy. A token with $K_t = 1$ does not contribute, and a token with a large K_t carries a symbol with $\log_2 K_t$ bits, so the encoder places checks at usable positions and adapts each check to the selected alphabet.

5.3 Undetectability and Correctness of CODETTA Keyed Protocol

We analyze undetectability, the substitution channel, and correctness and soundness of [Algorithm 2](#), following [Section 4](#). All three rest on [Lemma 1](#), the one-step lemma of the strawman, which applies unchanged. Part (i)

of the lemma is the single-step form of undetectability, which [Theorem 3](#) extends to whole transcripts and to pseudorandom masks, and parts (ii) and (iii) describe the errors that [Corollary 2](#) and [Theorem 4](#) count. The lemma fixes the complete state before step t , which includes the public prefix $V_{<t}$ and hence q_t , K_t , and the bucket splitting $B_{t,\cdot}$; its proof relies only on the partition being fixed and the symbol being uniform on $\mathbb{Z}_{K_t}$. It therefore holds at every usable step of [Algorithm 2](#) with $K = K_t$. As in the strawman, $Y_t \sim p_t$, the proposal is rejected with probability $\Delta_{K_t}(p_t)$, and the symbol is wrong exactly when the proposal is rejected. At a step with $K_t = 1$ the sender samples $Y_t \sim p_t$ directly, which is the lemma with a single bucket.

Theorem 3 (Undetectability of CODETTA). *If RangePRF is a secure pseudorandom function, CODETTA is computationally covert. Concretely, for every probabilistic polynomial-time distinguisher $\mathcal{A}$ in [Definition 2](#) there is a probabilistic polynomial-time distinguisher $\mathcal{A}_1$ against RangePRF such that $\text{Adv}_{\mathcal{A}}^{\text{steg}}(\kappa) \leq \text{Adv}_{\text{RangePRF}, \mathcal{A}_1}^{\text{prf}}(\kappa)$.*

Proof. We use a hybrid argument over three experiments.

- H_0 : the experiment of [Definition 2](#) with $b = 1$; every query is answered by EMBED of [Algorithm 2](#).
- H_1 : identical to H_0 , except that every mask $\text{RangePRF}_{k_m}(\text{sid}_i, s, K)$ is replaced by $F(\text{sid}_i, s, K)$ for a uniformly random function F whose value at an input with alphabet size K is uniform on $\mathbb{Z}_K$.
- H_2 : the experiment with $b = 0$; every query is answered by ordinary sampling from the base model without an embedded message.

Write $\mathbb{P}[H_j = 1]$ for the probability that $\mathcal{A}$ outputs 1 in H_j , so that $\text{Adv}_{\mathcal{A}}^{\text{steg}} = |\mathbb{P}[H_0 = 1] - \mathbb{P}[H_2 = 1]|$.

$H_0 \approx H_1$. Let $\mathcal{A}_1$ be the distinguisher against RangePRF that runs $\mathcal{A}$, simulates the challenger with the base model, the draft model, and the alphabet-adaptive ECC, obtains every mask by querying its oracle at (sid_i, s, K_t) , and outputs whatever $\mathcal{A}$ outputs. When the oracle is RangePRF_{k_m} , $\mathcal{A}_1$ simulates H_0 exactly; when it is a random function, $\mathcal{A}_1$ simulates H_1 exactly. Hence $|\mathbb{P}[H_0 = 1] - \mathbb{P}[H_1 = 1]| = \text{Adv}_{\text{RangePRF}, \mathcal{A}_1}^{\text{prf}}$, and $\mathcal{A}_1$ runs $\mathcal{A}$ once plus one protocol execution per query, so it is probabilistic polynomial-time whenever $\mathcal{A}$ is.

$H_1 \equiv H_2$. Fix a query (sid_i, m_i, T_i) and let $\mathcal{F}_{t-1}$ be the complete state before step t , consisting of the sender's state H_t , the transcript $V_{<t}$, the message, and all earlier randomness. We compare the conditional distribution of (Y_t, H_{t+1}) given $\mathcal{F}_{t-1}$ in the two experiments.

- The draft distribution q_t , the alphabet size K_t , and the partition $B_{t,\cdot}$ are deterministic functions of $V_{<t}$ and public parameters ([Algorithm 2](#), line 4), so they take the same values in both experiments and depend on the message only through the published prefix.
- If $K_t = 1$, the sender samples $Y_t \sim p_t$ in H_1 (line 6), which is the distribution of Y_t in H_2 by [Equation \(1\)](#).
- If $K_t > 1$, the check index s is determined by $V_{<t}$, and the pair (sid_i, s) never repeats across queries and steps, so $F(\text{sid}_i, s, K_t)$ is evaluated at a fresh point and is uniform on $\mathbb{Z}_{K_t}$, independent of $\mathcal{F}_{t-1}$ and of C_s . Hence $A_s = C_s + F(\text{sid}_i, s, K_t) \bmod K_t$ is uniform given $\mathcal{F}_{t-1}$, for every message, and [Lemma 1](#)(i) applied to the partition $B_{t,\cdot}$ gives $Y_t \mid \mathcal{F}_{t-1} \sim p_t$ in H_1 , which is again the distribution of Y_t in H_2 .
- After Y_t is published, the sender's state H_{t+1} is computed in the same way in both experiments; it is a function of H_t , $V_{\leq t}$, and fresh randomness of the agent, and it does not depend on the message, the mask, the token proposal, or whether the token proposal was accepted.

By induction on t , the distribution of the transcript returned to $\mathcal{A}$ is identical in H_1 and H_2 . So the view of $\mathcal{A}$ is identically distributed in the two experiments and $\mathbb{P}[H_1 = 1] = \mathbb{P}[H_2 = 1]$.

By the triangle inequality,

$$\text{Adv}_{\mathcal{A}}^{\text{steg}} \leq |\mathbb{P}[H_0 = 1] - \mathbb{P}[H_1 = 1]| + |\mathbb{P}[H_1 = 1] - \mathbb{P}[H_2 = 1]| = \text{Adv}_{\text{RangePRF}, \mathcal{A}_1}^{\text{prf}}.$$

If RangePRF is a secure PRF, the right side is negligible in κ for every probabilistic polynomial-time $\mathcal{A}$, so CODETTA is computationally covert. $\square$

Recall that check s is carried by token t_s , and let $M \leq T$ be the number of checks in the transcript. Let $E_s = \mathbf{1}\{\hat{C}_s \neq C_s\}$ indicate a substituted check and $W = \sum_{s \leq M} E_s$.

Corollary 2 (Variable-alphabet substitution channel). *The receiver observes exactly one check per usable token, and the usable steps and check indices are the same at both parties, so $\hat{C}_s$ and C_s refer to the same check for every s . Moreover, $\hat{C}_s \neq C_s$ if and only if the proposal at step t_s is rejected, and under uniform masks*

$$\mathbb{P}[E_s = 1 \mid \mathcal{F}_{t_s-1}] = \Delta_{K_{t_s}}(p_{t_s}) \leq \text{TV}(p_{t_s}, q_{t_s}) + \theta.$$

Proof. K_t and $f_{\text{Bucket},t}$ are deterministic functions of $V_{<t}$ and public parameters, and both parties condition on the same published prefix whether or not a proposal was rejected, so they compute the same alphabet size at every step and advance s at the same steps. A rejection changes Y_t and possibly $\hat{C}_s$, but K_t and s remain unchanged. Unmasking is a bijection of $\mathbb{Z}_{K_t}$, so $\hat{C}_s \neq C_s$ exactly when $f_{\text{Bucket},t}(Y_t) \neq A_s$, which by Lemma 1(iii) happens exactly when the proposal token is rejected by maximal coupling; by Lemma 1(ii) this has conditional probability $\Delta_{K_t}(p_t)$. The bound is Equation (13) with the constraint $\Delta_{K_t}(q_t) \leq \theta$. $\square$

Corollary 2 is the counterpart of Corollary 1. The channel is still a substitution channel, but its alphabet changes with the token, and its error rate is no longer the imbalance of a random partition. The public bucket partition imbalance is bounded by θ , and the remaining part is the mismatch between the sender's base model and the public draft model. In the symmetric case of $p_t = q_t$, the error rate is at most θ at every usable step.

Theorem 4 (Correctness and soundness of CODETTA). (i) *Correctness. Suppose the masks are uniform and $\text{TV}(p_t, q_t) \leq \varepsilon$ at every usable step, so that by Corollary 2 every check is substituted with conditional probability at most $\varepsilon + \theta$. For any decoder of the check sequence that recovers u whenever at most τ checks are substituted, the scheme is δ -correct with $\delta \leq \mathbb{P}[W > \tau]$, and for every $\tau > T(\varepsilon + \theta)$,*

$$\mathbb{P}[W > \tau] \leq \exp\left(-\frac{(\tau - T(\varepsilon + \theta))^2}{2T}\right). \quad (18)$$

(ii) *Soundness. If Tag is a secure pseudorandom function, the scheme is sound for every decoder. Concretely, if the receiver validates at most Q candidate messages, there is a probabilistic polynomial-time distinguisher $\mathcal{A}_2$ against Tag such that Extract returns a message other than m or $\perp$ with probability at most $Q2^{-\lambda} + \text{Adv}_{\text{Tag}, \mathcal{A}_2}^{\text{prf}}(\kappa)$.*

Proof. (i) *Correctness.* By Corollary 2, W counts the substituted checks, so a decoder that tolerates τ substitutions recovers u (m and its tag) whenever $W \leq \tau$. Hence, $\mathbb{P}[\text{Extract} \neq m] \leq \mathbb{P}[W > \tau]$. For the tail bound we need a martingale with a fixed number of increments, but checks are emitted only at usable steps and their number M is random. We therefore index by steps instead of by checks. For a usable step t , let E_t be the indicator that its check is substituted and $\Delta_t = \Delta_{K_t}(p_t)$ the conditional probability of this event; for a step with $K_t = 1$, which emits no check, set $E_t = \Delta_t = 0$. Then $W = \sum_{t \leq T} E_t$, and by Corollary 2 $\mathbb{P}[E_t = 1 \mid \mathcal{F}_{t-1}] = \Delta_t \leq \varepsilon + \theta$ at every step. Hence $Z_s = \sum_{t \leq s} (E_t - \Delta_t)$ is a martingale with increments in $[-1, 1]$, and $W = Z_T + \sum_{t \leq T} \Delta_t \leq Z_T + T(\varepsilon + \theta)$. Therefore $W > \tau$ implies $Z_T > \tau - T(\varepsilon + \theta)$, and the Azuma–Hoeffding inequality gives Equation (18).

(ii) *Soundness.* Let Bad be the event that a wrong candidate passes validation, that is, some $\hat{m}' \neq m$ whose decoded tag equals $\text{Tag}_{k_v}(\text{sid}, \hat{m}')$. Let $\mathcal{A}_2$ be the probabilistic polynomial-time distinguisher that runs the protocol with its oracle in place of Tag_{k_v} and outputs 1 if Bad occurs. Replacing Tag_{k_v} by a uniformly random function R onto $\{0, 1\}^\lambda$ changes $\mathbb{P}[\text{Bad}]$ by at most $\text{Adv}_{\text{Tag}, \mathcal{A}_2}^{\text{prf}}$, so it suffices to bound $\mathbb{P}[\text{Bad}]$ when R is used. In this case, embedding evaluates R only at (sid, m) , so the transcript, and hence every candidate produced from it, is independent of the values of R at all other points. Validating a candidate $\hat{m}' \neq m$ compares its decoded tag with $R(\text{sid}, \hat{m}')$, a fresh uniform value, so the comparison succeeds with probability $2^{-\lambda}$. At most Q distinct candidates are validated, and a union bound gives $\mathbb{P}[\text{Bad}] \leq Q2^{-\lambda}$. $\square$

Remarks. [Theorem 4\(i\)](#) covers bounded-distance decoders of the check sequence. CODETTA decodes with belief propagation, which offers no such guarantee, and [Section 7](#) reports its recovery rate on every workload instead. The tag separates the two possible outcomes of a decode. A failed decode returns $\perp$, and a wrong message passes validation only with the probability of [Theorem 4\(ii\)](#). The tail bound also makes the difference from [Theorem 2](#) visible. In the strawman, the compensator $\sum_t \Delta_K(p_t)$ runs over every token, including the near-noise ones; in CODETTA it runs over the usable tokens only, and every term is at most $\text{TV}(p_t, q_t) + \theta$.

A usable token with alphabet size K_t carries one symbol out of K_t , so it conveys at most $\log_2 K_t$ bits of payload under any ECC. The payload rate of a transcript therefore cannot exceed the average of $\log_2 K_t$ over its tokens. The delivered rate falls short of this ceiling by the cost of substitution errors, check redundancy, and the tag.

6 CODETTA Steganographic Key Exchange

The protocols so far assume that the sender and the receiver share a key $sk = (k_m, k_v)$. Independently deployed agents have no such key, especially if they are hosted by different organizations. They cannot establish the keys in the open, because a public key or a ciphertext in a conversation is an obvious signal to the auditor that they are transmitting secrets. The key exchange itself must therefore be hidden in ordinary text without a key.

Hiding without a key is much more challenging than keyed scenarios. In the keyed protocols, the mask is a pseudorandom pad that the auditor cannot compute without the key, while the sender and the receiver, who both hold the key, compute the same pad and cancel it. The pad is uniform to the auditor, and [Lemma 1](#) preserves the original distribution for every message. This construction does not generalize to the keyless setting.

One keyless way to make a token carry a symbol is a public hash. The sender draws candidate blocks of tokens from the base distribution and publishes one whose hash equals the symbol, and the receiver recomputes the hash of the published block to read the symbol. Every party can compute the hash, so every message is public. The transcript keeps the base distribution only if the symbols are uniform, because otherwise the published blocks will be biased. A recent work follows this route [[VZ26](#)] and hides a key exchange whose messages are pseudorandom, so that every symbol is uniform by construction. The cost of this route lies in how the exchange is carried. The labels of a finite candidate pool are never perfectly balanced, and a sender that keeps the transcript distribution exact has to publish a block with the wrong label with probability equal to this imbalance, so a published symbol is wrong with a constant probability. The key exchange of [[VZ26](#)] tolerates such noise at the level of key agreement, and the feedback compiler in its [Section 8](#) transmits the exchange bit by bit through a signaling protocol. Each published message carries a single bit of a key-exchange message, and the sender publishes a fixed number of messages for every bit, chosen in advance so that the receiver reads the bit correctly with high probability.

[Section 7.4](#) estimates the cost of their construction on our workloads and compares it with CODETTA. The same work also proves two impossibility results [[VZ26](#)]. First, a pseudorandom code whose decoder is public cannot correct a constant fraction of bit flips; our carrier is not such a code, because the receiver decodes with its MLWE secret and no public decoding information exists. Second, a non-interactive key exchange with a pseudorandom transcript, in which the two messages are generated non-adaptively and every transcript bit is flipped independently with a constant probability, admits a quasi-polynomial attack. Our exchange satisfies neither condition. The sender's response (w, z) depends on the share $\tilde{b}$ that it has read, so the messages are adaptive, and the carrier only displaces each q -ary coefficient by a small amount, whereas the result assumes that every transcript bit is flipped with a constant probability.

CODETTA instead transports lattice coefficients directly through the carrier, without a separate bitwise signaling layer, and lets the MLWE absorb the carrier's errors, and it certifies correctness on the workload itself ([Theorem 7](#)) instead of relying on the channel interface of their compiler, in which the sender knows

the crossover probability of each channel use before sending and learns afterwards whether the bit was flipped. The messages of MLWE key exchange are pseudorandom vectors of coefficients in $\mathbb{Z}_q$, and MLWE decryption tolerates small additive noise by design. We therefore send each coefficient as one q -ary symbol, about $\log_2 q \approx 11.7$ bits per carrier position, and we allow noise in the carrier channel (see [Section 6.4](#) for concrete parameters). The two designs thus take different directions from the same exact-distribution-preserving sampling. The construction of [\[VZ26\]](#) carries one bit per block and thereby works on short blocks of constant entropy, whereas CODETTA carries a full q -ary symbol per position and therefore requires enough entropy in every block ([Section 7.4](#) reports the failures on different workloads and models). A carrier error is a small displacement of a coefficient, which enters the exchange as extra noise and is absorbed together with the lattice noise, while a short ECC over the exchanged message corrects the residue. The carrier itself is exact. It publishes a block of text drawn from the base distribution at every position regardless of the symbol, so undetectability relies only on the pseudorandomness of the lattice messages. With the parameters of [Section 6.4](#), a full exchange of a 128-bit key takes 1,280 carrier positions. In the following, [Section 6.1](#) builds the carrier, [Section 6.2](#) runs the exchange over it, [Section 6.3](#) proves undetectability and correctness, and [Section 6.4](#) instantiates the concrete parameters.

6.1 Lifted Exact-Cover Carrier

The carrier builds on the keyless hash-based embedding of the previous paragraphs. Its unit is one carrier position, in which one party publishes one block of tokens and thereby transports one coefficient $a \in \mathbb{Z}_q$ of a lattice message; both messages of the exchange are sequences of such coefficients. We call the party that publishes the block the sender and the party that reads it the receiver; both agents take both roles during the exchange.

The naive embedding has two weaknesses. First, a block is published with a probability that depends on how many other blocks share its label, so for a fixed pool the published block is not a uniform member of the pool and is honest only after averaging over the hash values. Second, when the pool has no block with the target label, the fallback delivers a random value to the receiver, making the coefficient’s additive noise large. The carrier removes both weaknesses. Its selection rule makes the published block a uniform member of the pool for every fixed pool and label table, so the block is an honest sample regardless of the labels. When the pool has no block with the target label, the rule publishes a block whose label is close to the target, so an error is a small displacement of the target label. We first describe the candidate pool and the labels at one carrier position and then the selection rule.

Consider carrier position i , and let p_i be the sender’s honest distribution over its next response block given its own state and the transcript so far, where a block contains at most B tokens. We write $\mathcal{U}(S)$ for the uniform distribution on a finite set S . The sender draws a pool of $L = \beta q$ blocks

$$X_0, \dots, X_{L-1} \sim p_i \quad \text{independently,}$$

where $\beta \geq 1$ is a public replica ratio ([Algorithm 3](#), line 3). The pool is drawn by the sender’s model on its local context and does not depend on the target symbol a ; the pool is revealed neither to the public nor to the receiver. Every block receives a public label

$$y_j = \text{Hash}_q(\text{sid}, i, X_j) \in \mathbb{Z}_q,$$

where Hash_q is a hash with exact range $\mathbb{Z}_q$ whose input also includes a public session identifier sid , fresh for every exchange, and the position i , so that labels at different positions and in different exchanges are independent (line 4). Since the hash function is public, the receiver can compute the label of any published block (line 10).

If the labels were perfectly balanced, every value of $\mathbb{Z}_q$ would appear exactly β times in the pool, and the sender could publish a block with label a chosen uniformly among its β copies. However, the labels of

Algorithm 3 Lifted exact-cover carrier at one position

Require: modulus q ; public session identifier sid ; public replica ratio β and pool size $L = \beta q$; public serializer $g : \mathbb{Z}_q \rightarrow \mathcal{Y}$ with reconstruction map $g^+ : \mathcal{Y} \rightarrow \mathbb{Z}_q$ ($\mathcal{Y} = \mathbb{Z}_q$ and $g = g^+ = \text{id}$ for the identity serializer); public hash $\text{Hash}_{\mathcal{Y}}$ with exact range $\mathcal{Y}$

```
1: procedure PUBLISH( $i, a$ )
2:    $p_i \leftarrow$  the sender's distribution over its next block ▷ sender only
3:   sample  $X_0, \dots, X_{L-1} \sim p_i$  independently ▷ candidate pool, independent of  $a$ 
4:    $y_j \leftarrow \text{Hash}_{\mathcal{Y}}(\text{sid}, i, X_j)$  for  $j = 0, \dots, L-1$  ▷ public labels
5:   sort the slots  $(a', J) \in \mathbb{Z}_q \times \{0, \dots, \beta-1\}$  by target label  $g(a')$  and the blocks by label  $y_j$ 
6:    $\text{Assign} \leftarrow$  the alignment of the two sorted lists under the cyclic rotation of least total squared displacement (modulo  $|\mathcal{Y}|$ ) between  $g(a')$  and the label of the block at slot  $(a', J)$  ▷ bijection
7:   sample  $J \sim \mathcal{U}(\{0, \dots, \beta-1\})$ ; publish  $X_{\text{Assign}(a, J)}$  ▷ uniform over the pool
8: end procedure

9: procedure READ( $i, X$ )
10:   $y \leftarrow \text{Hash}_{\mathcal{Y}}(\text{sid}, i, X)$  ▷ recompute the label of the published block
11:  return  $g^+(y) \in \mathbb{Z}_q$  ▷ lift the label to a symbol
12: end procedure
```

a random pool are not balanced, and publishing only blocks with the target label would bias the transcript toward the labels that are overrepresented in the pool. The sender instead fixes a bijection

$$\text{Assign} : \mathbb{Z}_q \times \{0, \dots, \beta-1\} \longrightarrow \{0, \dots, L-1\}$$

between the L slots (a, J) and the L blocks, samples a replica $J \sim \mathcal{U}(\{0, \dots, \beta-1\})$, and publishes $X_{\text{Assign}(a, J)}$ (line 7). Because Assign is a bijection and (a, J) is uniform when a is, the published index is uniform over the pool for every fixed pool, so the published block is an honest sample from p_i with no constraint on the label. This selection rule follows the bundle sampler of [VZ26], which also publishes a uniformly chosen member of a pool of honest candidates, so that the marginal distribution stays exactly honest and the imbalance of the labels becomes a decoding error. The carrier extends it from a binary label to a q -ary symbol and chooses the bijection so that an error becomes a small displacement of the symbol instead of a flipped bit.

We call this construction a lifted exact cover. Every block is used by exactly one slot, which is the exact cover, and the replica index lifts the q labels to $L = \beta q$ slots. Undetectability holds for every choice of the bijection, because the argument above used only the fact that Assign is a bijection. The choice of Assign only determines the carrier error, and we choose it to keep the error small. Specifically, the sender sorts the blocks by label and the slots by symbol (Algorithm 3, line 5). It then aligns the two sorted lists and rotates one list against the other. Each rotation is a bijection, and its cost is the total squared displacement between the label of each block and the symbol of its slot. The sender keeps the rotation of least cost and reads Assign off the resulting alignment (line 6). The costs of all L rotations can be computed together by a cyclic correlation in $O(L \log L)$ time. When the labels are nearly balanced, almost every slot receives a block with its own label. The remaining slots receive a block whose label differs from the symbol by a small amount. A carrier error is therefore a small displacement of a modulo q instead of a random value.

For key exchange phases in which the receiver requires less precision, a public serializer $g : \mathbb{Z}_q \rightarrow \mathcal{Y}$ maps each symbol to a target label in a smaller alphabet $\mathcal{Y}$, and a public reconstruction map $g^+ : \mathcal{Y} \rightarrow \mathbb{Z}_q$ lifts a received label back to a symbol. In such a phase, the blocks are hashed into $\mathcal{Y}$ instead of $\mathbb{Z}_q$, the slot (a, J) is matched to the target label $g(a)$, and the receiver returns $g^+(y)$ for the label y it reads (Algorithm 3,

Algorithm 4 Three-phase key exchange over the lifted exact-cover carrier

Public parameters. Session identifier sid , matrix $A \in R_q^{k \times k}$, distributions χ_{sec} and χ_{err} , ECC (ECC, DecECC) with codeword length N , serializers $g_b = \text{id}$, g_w , g_z , and KDF. Every arrow sends one coefficient per carrier position through [Algorithm 3](#), and V is the public transcript of the three phases.

Sender	Receiver
<i>Setup</i>	
1 $s_S \sim \chi_{\text{sec}}^k, e_1 \sim \chi_{\text{err}}^k, e_2 \sim \chi_{\text{err}}$	$s_R \sim \chi_{\text{sec}}^k, e_R \sim \chi_{\text{sec}}^k$
<i>Phase 1, the receiver's share</i>	
2	$b \leftarrow As_R + e_R \in R_q^k$
3 read $\tilde{b} \in R_q^k$	$\xleftarrow[kd \text{ positions, } g_b]{b}$
<i>Phases 2 and 3, the sender's response</i>	
4 $K_0 \sim \mathcal{U}(\{0, 1\}^\ell); c^+ \leftarrow \text{ECC}(K_0) \parallel 0^{d-N}$	
5 $w \leftarrow A^T s_S + e_1 \in R_q^k$	
6 $z \leftarrow \tilde{b}^T s_S + e_2 + \iota(c^+) \in R_q$	
7	$\xrightarrow[kd \text{ positions, } g_w]{w}$ read $\tilde{w} \in R_q^k$
8	$\xrightarrow[d \text{ positions, } g_z]{z}$ read $\tilde{z} \in R_q$
9	$\omega \leftarrow \tilde{z} - s_R^T \tilde{w}$
10	$\hat{c} \leftarrow \text{HD}(\omega)_{[N]}; \hat{K}_0 \leftarrow \text{DecECC}(\hat{c})$
<i>Key derivation and confirmation</i>	
11 $K_S \leftarrow \text{KDF}(K_0, V)$	$K_R \leftarrow \text{KDF}(\hat{K}_0, V)$
12 embed a fixed message under K_S (Algorithm 2)	$\xrightarrow{\text{keyed protocol}}$ validate under K_R , else restart

line 11). The slots remain indexed by the uncompressed symbol $a \in \mathbb{Z}_q$, so the bijection and the published distribution are unchanged.

6.2 Key Exchange

The exchange transports a random pre-key from one party to the other over the carrier of [Section 6.1](#), and both parties derive the session key from the transmitted pre-key. We name the two parties by their roles in the keyed protocol that follows. The sender chooses the pre-key and later embeds messages under the resulting key, and the receiver recovers the pre-key and later extracts those messages. During the exchange, each party embeds the pseudorandom message into its own responses. The receiver publishes a pseudorandom share first, and the sender then publishes a vector and the masked pre-key, so the key exchange has three phases. [Algorithm 4](#) shows the three phases with the sender on the left and the receiver on the right, and numbers the steps that the following paragraphs refer to.

The key exchange should fit the carrier of [Section 6.1](#), which restricts the messages it transports in two ways. First, every transported symbol must be uniform on $\mathbb{Z}_q$, since the published block is an honest sample only when the symbol is uniform ([Lemma 2](#)). Second, the exchange must tolerate a small error in every symbol, since the carrier can deliver a label near the target. A key transport based on MLWE satisfies both requirements by design. Under the MLWE assumption, every message of the transport is computationally indistinguishable from a uniform vector over $\mathbb{Z}_q$, so each of its coefficients can be handed to the carrier as a uniform q -ary symbol. The transport is also robust to small additive errors, because the receiver recovers the

payload by rounding each coefficient in the decoding to the nearer of two values that are $q/2$ apart, and an error that is small compared with this distance does not change the rounding. A displacement introduced by the carrier therefore has the same effect as the lattice noise that the transport already tolerates, and the two can be analyzed together.

We now describe the exchange step by step, following [Algorithm 4](#).

Setup. Let $R_q = \mathbb{Z}_q[x]/(x^d + 1)$ be the polynomial ring of degree d , and let k be the module rank. An element of R_q^k has kd coefficients in $\mathbb{Z}_q$ and occupies kd carrier positions. Let $A \in R_q^{k \times k}$ be a public random matrix, and let sid be a public session identifier that is fresh for every exchange and enters every hash of the carrier. Before the exchange starts, the receiver samples a secret $s_R \sim \chi_{\text{sec}}^k$ and an error $e_R \sim \chi_{\text{err}}^k$, and the sender samples a secret $s_S \sim \chi_{\text{sec}}^k$ and errors $e_1 \sim \chi_{\text{err}}^k$ and $e_2 \sim \chi_{\text{err}}^k$ (step 1). Here χ_{sec} and χ_{err} are distributions over polynomials with small coefficients; see [Section 6.4](#). The exchange sends three messages through the carrier, including the receiver's share $b \in R_q^k$, the sender's vector $w \in R_q^k$, and the masked pre-key $z \in R_q$, which occupy kd , kd , and d carrier positions. A party reads each message with a small displacement, and we write the values it reads as $\tilde{b} \in R_q^k$, $\tilde{w} \in R_q^k$, and $\tilde{z} \in R_q$, and the displacements as $\zeta_b = \tilde{b} - b$, $\zeta_w = \tilde{w} - w$, and $\zeta_z = \tilde{z} - z$.

The receiver's share. In the first phase, the receiver computes

$$b = As_R + e_R \in R_q^k$$

and publishes its kd coefficients, one per carrier position, with the identity serializer (step 2). The identity serializer is required in this phase, since the security proof needs the value that the sender reads to be uniform on R_q^k ([Lemma 3](#)). The share is an MLWE sample under the receiver's secret, so under the MLWE assumption it is computationally indistinguishable from uniform and can be handed to the carrier as a sequence of uniform symbols ([Section 6.3](#)). The sender reads $\tilde{b} = b + \zeta_b$ (step 3).

The sender's response. The sender samples a pre-key $K_0 \sim \mathcal{U}(\{0, 1\}^\ell)$ and encodes it with an ECC that corrects every error pattern of weight at most τ ,

$$c = \text{ECC}(K_0) \in \{0, 1\}^N, \quad N \leq d.$$

It appends $d - N$ zero bits to obtain $c^+ \in \{0, 1\}^d$ and maps c^+ to the ring element $\iota(c^+) \in R_q$, where ι places a 0 bit at coefficient 0 and a 1 bit at coefficient $\lfloor q/2 \rfloor$ (step 4). The sender then computes (steps 5 and 6)

$$w = A^T s_S + e_1 \in R_q^k, \tag{19}$$

$$z = \tilde{b}^T s_S + e_2 + \iota(c^+) \in R_q, \tag{20}$$

and publishes, in the second and third phases, the kd coefficients of w with serializer g_w and the d coefficients of z with serializer g_z (steps 7 and 8). The vector w is an MLWE sample under the sender's secret and plays the same role for the sender as the share b plays for the receiver. The masked pre-key z hides the encoded pre-key $\iota(c^+)$ under the mask $\tilde{b}^T s_S + e_2$, which the sender computes from its own secret and the share it read. [Lemma 3](#) proves that the coefficients of b , w , and z handed to the carrier are jointly pseudorandom, so the carrier transports them as uniform symbols.

Receiver decoding. The receiver reads $\tilde{w} = w + \zeta_w$ and $\tilde{z} = z + \zeta_z$ through the reconstruction maps g_w^+ and g_z^+ (steps 7 and 8) and computes in step 9:

$$\omega = \tilde{z} - s_R^T \tilde{w} \pmod{q}.$$

The following expansion shows that the pre-key K_0 can be obtained from ω . $\tilde{z}$ equals:

$$\tilde{z} = \tilde{b}^T s_S + e_2 + \iota(c^+) + \zeta_z = s_R^T A^T s_S + e_R^T s_S + \zeta_b^T s_S + e_2 + \zeta_z + \iota(c^+),$$

and the value that the receiver subtracts equals:

$$s_R^T \tilde{w} = s_R^T A^T s_S + s_R^T e_1 + s_R^T \zeta_w.$$

Both contain the ring element $s_R^T A^T s_S$, which combines the two secrets through the public matrix A , and this common term cancels in the subtraction, which leaves:

$$\omega = \iota(c^+) + \underbrace{e_R^T s_S + \zeta_b^T s_S + e_2 + \zeta_z - s_R^T e_1 - s_R^T \zeta_w}_{\text{noise}} \pmod{q}.$$

Every noise term is either a small error or displacement or the product of one with a small secret, so the noise is small compared with $q/2$ in almost every coefficient. The receiver applies the hard-decision map $\text{HD} : R_q \rightarrow \{0, 1\}^d$, which rounds each coefficient to the nearer of 0 and $\lfloor q/2 \rfloor$, keeps the first N bits as $\hat{c} = \text{HD}(\omega)_{[N]}$, and decodes $\hat{K}_0 = \text{DecECC}(\hat{c})$ (step 10). A coefficient is decided wrongly only where the noise reaches about $q/4$, and the ECC corrects up to τ such coefficients. [Theorem 6](#) states this decoding condition exactly.

Key derivation. In step 11, both parties derive the session key from the pre-key and the public transcript V of the exchange,

$$K_S = \text{KDF}(K_0, V) \in \{0, 1\}^\kappa, \quad K_R = \text{KDF}(\hat{K}_0, V) \in \{0, 1\}^\kappa,$$

and the two keys agree whenever the receiver has decoded the pre-key K_0 correctly.

Key confirmation. The two parties then run the keyed protocol of [Section 5](#) under their keys to transmit a fixed public message (step 12). If the two keys agree, the receiver extracts the message and its tag validates. If they disagree, validation fails, and the parties repeat the exchange with fresh randomness. Each attempt fails independently with some probability p_{fail} , so the probability that r independent attempts all fail is $\leq p_{\text{fail}}^r$. For example, an exchange that succeeds with probability 0.99 fails three attempts with a probability $\leq 10^{-6}$.

6.3 Undetectability and Correctness of CODETTA Steganographic Key Exchange

The auditor of this section is passive. It reads every public message and may query the carrier hash and the KDF, which we model as random oracles, but it cannot modify, delay, inject, or choose messages. Since the two parties never see the auditor's queries, an auditor that queries the oracles during the exchange has the same view as one that makes the same queries after the transcript is complete, so we analyze the latter.

Lemma 2 (Labeled lifted exact cover). *Fix one position, candidate pool, label table, serializer, and bijection Assign. If $(a, J) \sim \mathcal{U}(\mathbb{Z}_q \times \{0, \dots, \beta - 1\})$, then $\text{Assign}(a, J)$ is uniform over the L candidate indices, and after averaging over the pool the published block has exactly the distribution p_i . If the symbols of all positions are independent and uniform, and independent of the pools, then the published blocks of all positions are jointly distributed as an honest transcript. For the identity serializer, the published block X and its label y are jointly distributed as $X \sim p_i$ and $y \sim \mathcal{U}(\mathbb{Z}_q)$ with X and y independent, so the value that the reader of the position obtains is uniform on $\mathbb{Z}_q$.*

Proof. Fix the pool $X_0, \dots, X_{L-1}$ and its label table, which fixes the bijection Assign. The pair (a, J) is uniform over a set of size L , and Assign maps this set bijectively onto the L candidate indices. Hence the published index $I = \text{Assign}(a, J)$ is uniform over the indices. This holds for every pool and label table, so I is independent of the pool. Now let the pool vary. Its blocks are independent samples from p_i , and I is uniform and independent of them, so the published block $X = X_I$ has distribution p_i . Duplicate blocks and unbalanced labels do not change this indices-level argument. The composition over positions follows from the chain rule. Condition on the public history before position i , which includes all earlier published blocks. Given this history, the pool at position i consists of independent samples from p_i , and the symbol is uniform

and independent of the pool by assumption. The argument above then shows that the published block at position i has distribution p_i given the history. An honest transcript is generated by exactly these conditional distributions, so the joint distribution of all published blocks equals that of an honest transcript. For the identity serializer, the label of the published block is $y = \text{Hash}_q(\text{sid}, i, X)$. In the random oracle model, the value of the hash at the fresh input (sid, i, X) is uniform on $\mathbb{Z}_q$ for every fixed X , so y is uniform and independent of X . Note that y differs from the symbol a by the displacement of the carrier, and this displacement is correlated with a . The uniformity of y does not rely on the displacement, but only on y being the hash value at a uniformly selected block. Inputs at different positions and in different exchanges are distinct, so their labels are independent. With the identity serializer, the reader obtains y itself. $\square$

Lemma 3 uses the last statement of the lemma for the first phase, in which the sender reads the share $\tilde{b}$ and uses it as a row of an MLWE matrix. For a lossy serializer, the label is uniform on the smaller alphabet $\mathcal{Y}$, but the lifted value $g^+(y)$ is not uniform on $\mathbb{Z}_q$, which is why the first phase uses the identity serializer.

Uniformity in the lemma is over the randomness of the random oracle, and a label that is predictable because its block is likely under p_i is compatible with the argument, since the MLWE assumption hands the matrix M to the distinguisher anyway.

Let $\text{Adv}_{k, \mathcal{B}}^{\text{mlwe}}(\kappa)$ be the advantage of a distinguisher $\mathcal{B}$ in telling $(A, A s_R + e_R)$ from (A, U) for $A \sim \mathcal{U}(R_q^{k \times k})$ and $U \sim \mathcal{U}(R_q^k)$, and let $\text{Adv}_{k+1, \mathcal{B}}^{\text{mlwe}}(\kappa)$ be its advantage in telling $(M, M s_S + e)$ from (M, U') for $M \sim \mathcal{U}(R_q^{(k+1) \times k})$, $e = (e_1, e_2)$, and $U' \sim \mathcal{U}(R_q^{k+1})$. These are the two decisional MLWE assumptions behind the receiver's share and the sender's response. For a passive auditor $\mathcal{A}$, let $\text{Adv}_{\mathcal{A}}^{\text{cover}}(\kappa)$ be its advantage in distinguishing the public transcript from an honest transcript, and $\text{Adv}_{\mathcal{A}}^{\text{key}}(\kappa)$ its advantage in additionally distinguishing the sender's key from an independent uniform κ -bit string. For a distribution $\mathcal{D}$ over complete sessions, let $\delta_{\mathcal{D}} = \mathbb{P}_{\Omega \sim \mathcal{D}}[K_R \neq K_S]$ be the probability that the two parties end with different keys.

Lemma 3 (Joint pseudorandomness of the transported symbols). *Consider the experiment in which a passive auditor reads the completed transcript of the exchange, and the modified experiment in which the carrier is fed independent uniform symbols at every position of all three phases instead of the coefficients of b , w , and z . For every probabilistic polynomial-time auditor $\mathcal{A}$ there are probabilistic polynomial-time distinguishers $\mathcal{B}_1$ and $\mathcal{B}_2$ such that $\mathcal{A}$ distinguishes the two experiments with advantage at most $\text{Adv}_{k, \mathcal{B}_1}^{\text{mlwe}}(\kappa) + \text{Adv}_{k+1, \mathcal{B}_2}^{\text{mlwe}}(\kappa)$.*

Proof. We use a hybrid argument over three experiments, where the auditor reads the completed transcript.

- H_0 : the real exchange of **Algorithm 4**.
- H_1 : identical to H_0 , except that the receiver feeds the carrier a uniform vector $u_b \sim \mathcal{U}(R_q^k)$ in place of $b = A s_R + e_R$; the sender is unchanged and computes w and z from the labels $\tilde{b}$ it reads.
- H_2 : identical to H_1 , except that the sender feeds the carrier a uniform pair $(u_w, u_z) \sim \mathcal{U}(R_q^k \times R_q)$ in place of (w, z) ; this is the modified experiment of the lemma.

Write $\mathbb{P}[H_j = 1]$ for the probability that $\mathcal{A}$ outputs 1 in H_j .

$H_0 \approx H_1$. Let $\mathcal{B}_1$ be the distinguisher that receives (A, t) with $t \in R_q^k$, runs the exchange with A as the public matrix and the coefficients of t as the receiver's symbols, simulates the sender honestly, runs $\mathcal{A}$ on the resulting transcript, and outputs the output of $\mathcal{A}$. When $t = A s_R + e_R$, $\mathcal{B}_1$ simulates H_0 exactly; when t is uniform, it simulates H_1 exactly. Hence

$$|\mathbb{P}[H_0 = 1] - \mathbb{P}[H_1 = 1]| = \text{Adv}_{k, \mathcal{B}_1}^{\text{mlwe}}.$$

$H_1 \approx H_2$. In H_1 , the symbols of the first phase are uniform, so by **Lemma 2** the published blocks of the first phase are honest samples and the vector $\tilde{b}$ of their labels is uniform on R_q^k and independent of the blocks and of A . Here $\tilde{b}$ is the vector of hash values $\text{Hash}_q(\text{sid}, i, X_i)$ of the published blocks, and its uniformity

comes from the hash; the carrier displacement $\tilde{b} - u_b$ is correlated with u_b , but u_b enters H_1 only through the selection of the blocks. Let

$$M = \begin{pmatrix} A^T \\ \tilde{b}^T \end{pmatrix} \in R_q^{(k+1) \times k}, \quad e = (e_1, e_2) \in R_q^{k+1},$$

so that M is uniform and the sender's symbols in H_1 are

$$(w, z) = Ms_S + e + (0, \iota(c^+)).$$

In H_2 the sender's symbols are (u_w, u_z) , and $(u_w, u_z) - (0, \iota(c^+))$ is again uniform, so the two experiments differ exactly in whether $Ms_S + e$ or a uniform element of R_q^{k+1} is added to $(0, \iota(c^+))$. Let $\mathcal{B}_2$ be the distinguisher that receives (M, t') with $t' \in R_q^{k+1}$ and proceeds as follows.

- (a) It uses the first k rows of M as A^T and reads the last row as $\tilde{b} \in R_q^k$.
- (b) For each position i of the first phase, it samples the pool honestly, publishes a uniformly chosen pool member X_i , and computes $\text{Hash}_q(\text{sid}, i, X_i)$ as the coefficient of $\tilde{b}$ that position i carries. By [Lemma 2](#), the published block and its label have exactly their joint distribution in H_1 , and no earlier oracle query constrains the computed hash value.
- (c) It runs the second and third phases of the carrier with the symbols $t' + (0, \iota(c^+))$, runs $\mathcal{A}$ on the completed transcript, and outputs the output of $\mathcal{A}$.

When $t' = Ms_S + e$, $\mathcal{B}_2$ simulates H_1 exactly; when t' is uniform, it simulates H_2 exactly. Hence

$$|\mathbb{P}[H_1 = 1] - \mathbb{P}[H_2 = 1]| = \text{Adv}_{k+1, \mathcal{B}_2}^{\text{mlwe}}.$$

By the triangle inequality,

$$|\mathbb{P}[H_0 = 1] - \mathbb{P}[H_2 = 1]| \leq \text{Adv}_{k, \mathcal{B}_1}^{\text{mlwe}} + \text{Adv}_{k+1, \mathcal{B}_2}^{\text{mlwe}},$$

and both distinguishers run $\mathcal{A}$ once plus one execution of the exchange, so they are probabilistic polynomial-time whenever $\mathcal{A}$ is. $\square$

Theorem 5 (Undetectability and key security). *For every probabilistic polynomial-time passive auditor $\mathcal{A}$ making at most Q_{KDF} KDF queries, there are probabilistic polynomial-time distinguishers $\mathcal{B}_1$ and $\mathcal{B}_2$ such that*

$$\text{Adv}_{\mathcal{A}}^{\text{cover}}(\kappa) \leq \text{Adv}_{k, \mathcal{B}_1}^{\text{mlwe}}(\kappa) + \text{Adv}_{k+1, \mathcal{B}_2}^{\text{mlwe}}(\kappa), \quad (21)$$

$$\text{Adv}_{\mathcal{A}}^{\text{key}}(\kappa) \leq \text{Adv}_{k, \mathcal{B}_1}^{\text{mlwe}}(\kappa) + \text{Adv}_{k+1, \mathcal{B}_2}^{\text{mlwe}}(\kappa) + \frac{Q_{\text{KDF}}}{2^\ell}. \quad (22)$$

Proof. By [Lemma 3](#), the auditor's view is indistinguishable, up to $\text{Adv}_{k, \mathcal{B}_1}^{\text{mlwe}} + \text{Adv}_{k+1, \mathcal{B}_2}^{\text{mlwe}}$, from the transcript of an exchange whose carrier is fed independent uniform symbols at every position. By [Lemma 2](#), composed over positions and applied with slots indexed by $\mathbb{Z}_q$ regardless of the serializers, that transcript consists of honest samples, which proves [Equation \(21\)](#).

In the modified experiment, the transcript is independent of the pre-key K_0 , so the sender's key $\text{KDF}(K_0, V)$ is uniform unless the auditor queries the KDF at the point (K_0, V) , which happens with probability at most $Q_{\text{KDF}}/2^\ell$, and this proves [Equation \(22\)](#). $\square$

Correctness relies on a deterministic decoding region. The next theorem does not assume any model of the carrier errors and a concrete workload determines how often an execution leaves the region.

Theorem 6 (Noise-resilient decoding). *For any realized execution, write $\tilde{b} = b + \zeta_b$, $\tilde{w} = w + \zeta_w$, and $\tilde{z} = z + \zeta_z$ modulo q , where ζ_w and ζ_z include both the serializer reconstruction error and the carrier displacement. The receiver's residual satisfies*

$$\omega = \iota(c^+) + e_R^T s_S + \zeta_b^T s_S + e_2 - s_R^T e_1 - s_R^T \zeta_w + \zeta_z \pmod{q}. \quad (23)$$

Let $\hat{c} = \text{HD}(\omega)_{[N]}$ and $W = d_H(\hat{c}, c)$. If $W \leq \tau$, then the decoder recovers K_0 and $K_R = K_S$, for arbitrary values of $(\zeta_b, \zeta_w, \zeta_z)$.

Proof. Substitute $b = A s_R + e_R$, $w = A^T s_S + e_1$, and $z = \tilde{b}^T s_S + e_2 + \iota(c^+)$ into $\omega = \tilde{z} - s_R^T \tilde{w}$. The two copies of $s_R^T A^T s_S$ cancel, which yields Equation (23). The ECC corrects every error pattern of weight at most τ , so the recovered pre-key and the transcript agree at both parties, and so do their KDF outputs. $\square$

Equation (23) separates the noise of the lattice scheme from the noise of the carrier. The lattice noise terms $e_R^T s_S$, e_2 , and $s_R^T e_1$ are in the MLWE schemes, and the carrier adds $\zeta_b^T s_S$, $s_R^T \zeta_w$, and ζ_z . Because s_R and s_S are small, a displacement of a few units in a coefficient of b or w adds only a few units to the residual, and a displacement of z enters directly. The hard decision is wrong only in the coefficients where these terms accumulate to about $q/4$, and the ECC corrects up to τ such coefficients.

Corollary 3 (Workload-specific correctness). *For every distribution $\mathcal{D}$ over the complete sessions, $\delta_{\mathcal{D}} \leq \mathbb{P}_{\Omega \sim \mathcal{D}}[W > \tau]$.*

Since the displacement depends on the language model and the task, we certify correctness empirically.

Theorem 7 (Empirical certification). *Let $\Omega_1, \dots, \Omega_m \sim \mathcal{D}$ be independent complete sessions, let W_i be the value of W in session Ω_i , let $F_i = \mathbf{1}\{W_i > \tau\}$, and let $S = \sum_{i=1}^m F_i$. For a confidence level $1 - \alpha$ and $S < m$, with probability at least $1 - \alpha$ over the evaluation sample,*

$$\delta_{\mathcal{D}} \leq \mathbb{P}_{\mathcal{D}}[W > \tau] \leq \text{Beta}^{-1}(1 - \alpha; S + 1, m - S),$$

and for $S = 0$ the bound is $\delta_{\mathcal{D}} \leq 1 - \alpha^{1/m}$.

Proof. The F_i are independent Bernoulli variables with mean $\mathbb{P}_{\mathcal{D}}[W > \tau]$, the bound is the exact one-sided Clopper–Pearson bound for their mean, and Corollary 3 transfers it to key disagreement. $\square$

6.4 Instantiation

The proofs above hold for any choice of the lattice parameters, the serializers, and the ECC. This section fixes the choice used in the evaluation and states the security level it provides. The engineering parameters of the carrier, such as the replica ratios and the block length, do not enter the proofs, and Section 7 gives them with the experimental setup.

Lattice parameters. We use the ring R_q with $d = 256$ and $q = 3329$, rank $k = 2$, $\chi_{\text{sec}} = \text{CBD}_3$, and $\chi_{\text{err}} = \text{CBD}_2$, where CBD_{η} is the centered binomial distribution of parameter η . These are the parameters of ML-KEM-512 [Nat24]. The two assumptions of Theorem 5, decisional MLWE with k samples for the share and with $k + 1$ samples for the response, are the two assumptions in the IND-CPA security proof of ML-KEM-512, at the same ring, modulus, rank, and noise distributions [Nat24]. The exchange therefore relies on the same MLWE problems as ML-KEM-512, which NIST places in security category 1, at least as hard to break as exhaustive key search on AES-128.

Serializers. The share b is sent with the identity serializer, as Lemma 3 requires. The response uses $g_w = \text{Compress}_{10}$ and $g_z = \text{Compress}_4$, the ciphertext compression of ML-KEM-512, and the hard-decision map

HD is Compress_1 . Compression to 10 bits changes a coefficient by at most $q/2^{11} \approx 1.6$ and compression to 4 bits by at most $q/2^5 \approx 104$, and both are well below the decision threshold of $q/4 \approx 832$ in Equation (23).

Pre-key and ECC. The pre-key has $\ell = 128$ bits, which matches the security category and makes the term $Q_{\text{KDF}}/2^\ell$ in Equation (22) negligible at the same level, and the session key has $\kappa = 256$ bits. The ECC must place its codeword in the $d = 256$ coefficients of one ring element and carry at least ℓ information bits, and within these two constraints we take the largest correction radius. The binary BCH code of length 255 and dimension 131 of Section 2.4 corrects $\tau = 18$ errors and meets both constraints. Its 131 information bits are the pre-key followed by three zero bits, and the unused coefficient of the ring element is set to zero. If the hard decisions were wrong independently with probability p per coefficient, this code would fail with probability at most 10^{-6} for every $p \leq 0.0196$. Section 7 reports the measured failure rates. The exchange occupies $kd + kd + d = 1,280$ carrier positions.

7 Evaluation

7.1 Experimental setup

Tasks. We evaluate on three simulated agent workloads. FINANCIAL is a synthetic issuer-intelligence task with 20 issuers and 100 database records: the receiver agent asks about an issuer’s financial information, and the sender agent answers from its internal database. IFEVAL [ZLM⁺23] consists of natural-language requests with constraints on length, structure, wording, and formatting. CTF uses the offline InterCode-CTF [YPNY23] adaptation of picoCTF [CBB14]. For each CTF challenge, an agent first solves the challenge with shell and Python tools in a sandbox, and the carrier is the technical solution report that the same model then writes from its complete solve trajectory. These tasks simulate the realistic scenarios we have introduced in Section 1. In FINANCIAL and IFEVAL, the agents communicate in a conversational setup, and in CTF, the agents communicate by reading and writing to a shared file system.

Asymmetry. The sender agent runs a large base model on its full private context; the receiver sees only public transcript, including its own request, the sender’s response, and public problem statement. On FINANCIAL, the private context is the retrieved records. On IFEVAL, the sender first drafts a plan in its thinking trace, and that trace is the private context. On CTF, the private context is the agent’s full problem solving trajectory, including the task files, the tool commands, and the tool outputs. The receiver does not access the sender’s model, internal state, or private context. A draft-assisted receiver runs a small public draft model on the visible context alone; a model-free receiver does not run a language model.

Models. For sender models, we evaluate on widely used open-sourced models, including Qwen2.5-7B [YYZ⁺24], Llama-3.1-8B [GDJ⁺24], and Ministral-3-8B [LKS⁺26]; the public draft models are Qwen2.5-1.5B, Llama-3.2-1B, and Ministral-3-3B. The model-mismatch ablation below pairs them across families, and the draft-size ablation additionally uses Qwen2.5-0.5B, 3B, and 7B as drafts.

Baselines & Settings. ARS [BYP⁺25] is the state-of-the-art practical steganographic protocol under asymmetric resources, and we use its official implementation. We also construct a second baseline, HARQ LDPC, which replaces our alphabet-adaptive ECC with a rateless LDPC code: a 5G NR mother code of rate 0.2 whose parity bits are sent incrementally until the receiver decodes. We evaluate it in a draft-assisted and a draft-free configuration, using the optimized implementation in NVIDIA Sionna [HCAA⁺22].

CODETTA and the LDPC baselines carry 1,024 payload bits per message and append a 32-bit tag, so each message is 1,056 bits. The official ARS implementation delivers far fewer bits per token, so we give it a 128-bit payload on FINANCIAL and IFEVAL and a 16-bit payload on CTF; a shorter payload only helps ARS in a capacity comparison. We evaluate the average capacity of 25 independent secret messages with the same payload seed. A secret message spans several responses, i.e., the sender keeps embedding it across

consecutive responses until the receiver decodes it, and when a workload runs out of requests we rotate through its request pool again. Our primary metric is the end-to-end delivered payload rate (capacity)

$$R_{\text{payload}} = \frac{\text{correctly recovered payload bits}}{\text{effective visible carrier tokens}}.$$

We also report encoding and decoding speed in tokens per second. We run the experiments on one NVIDIA H200 GPU.

Ablations. To better understand the effectiveness of different components, we conduct three ablation studies. First, we vary information asymmetry by revealing 0%, 25%, 50%, 75%, or 100% of the private context to the receiver. Second, we vary model mismatch across same-model, cross-size, and cross-family sender–draft pairs. Third, we remove one component of our code at a time by comparing with LDPC baselines: dropping the draft model removes public channel estimation, and fixing the alphabet size removes adaptive alphabet size selection, so the two variants separate the capacity gained from each.

Key exchange. We run the key exchange of [Section 6](#) with the three sender models on FINANCIAL, IFEVAL, and CTF. Each of the $m = 100,000$ sessions replays the exchange algebraically, with fresh lattice randomness and with displacements drawn independently for every position from the distribution measured on the sender model’s candidate pools under eight independent session identifiers, so [Theorem 7](#) certifies the failure probability under this measured displacement distribution. The carrier uses the first complete response block of a turn as its token block, capped at $B = 64$ tokens, and the replica ratios are $(\beta_b, \beta_w, \beta_z) = (4, 4, 1)$. The Compress₄ phase has only 16 labels, so one replica suffices there.

7.2 Capacity of keyed protocols

[Table 1](#) reports the payload rate of the three methods on the three workloads and the three model pairs. Every cell recovered all 25 messages. CODETTA delivers between 4.1 and 6.4 times the rate of the draft-free LDPC baseline and between 16 and 94 times the rate of ARS. The gain over the draft-free baseline is consistent across models and workloads, and [Section 7.3](#) separates the two components behind it. The absolute rates differ by an order of magnitude across settings because they follow the entropy of the carrier text. Ministral writes the most varied text and reaches 0.45 to 0.57 bits per token on FINANCIAL and IFEVAL. Qwen on CTF is the other extreme. Its reports copy long spans of the private solve trajectory verbatim, 60% of the body tokens have an entropy below 0.1 bits, and every method drops below 0.04 bits per token there.

Task quality. [Theorem 3](#) guarantees that the covert transcripts follow the sender’s own distribution, so we do not measure their perplexity and instead check the task quality directly on IFEVAL, whose scorer is automatic. Ordinary responses of Qwen, Llama, and Ministral reach a strict instruction-following accuracy of 0.795, 0.795, and 0.744 on the evaluation prompts, and the covert transcripts of CODETTA reach 0.718, 0.795, and 0.615. No method differs significantly from ordinary generation under a two-sided exact McNemar test over the 25 messages, with a smallest p -value of 0.125.

Speed. Embedding adds a draft forward pass and the coupling step to every token, and the throughput rows of [Table 1](#) present this cost. The CODETTA sender runs at 82% to 103% of the speed of the ARS sender across the different models and tasks, and at 88% or higher in all but one, because generation time is dominated by the base model and the draft is much smaller. The draft-free LDPC sender is the fastest, since it samples from the base model with no draft at all.

For decoding cost, the two baselines decode with hashes alone and reach thousands to hundreds of thousands of tokens per second, while the CODETTA receiver replays the draft model over the transcript and decodes at 240 to 1,260 tokens per second, still 2 to 6 times faster than the sender generates.

Method	FINANCIAL			IFEVAL			CTF		
	Qwen	Llama	Ministral	Qwen	Llama	Ministral	Qwen	Llama	Ministral
<i>Capacity (bits per token)</i>									
CODETTA	0.1354	0.2289	0.4476	0.2510	0.3544	0.5685	0.0303	0.2844	0.2511
HARQ LDPC, draft-free	0.0280	0.0358	0.0936	0.0528	0.0665	0.1378	0.0072	0.0465	0.0571
ARS	0.0016	0.0142	0.0131	0.0044	0.0189	0.0204	0.0003	0.0064	0.0064
<i>Gain of CODETTA</i>									
over HARQ LDPC	4.8×	6.4×	4.8×	4.7×	5.3×	4.1×	4.2×	6.1×	4.4×
over ARS	87×	16×	34×	57×	19×	28×	94×	44×	39×
<i>Encoding throughput (tokens per second)</i>									
CODETTA	244	204	155	212	350	152	72	85	56
HARQ LDPC, draft-free	306	249	203	331	472	205	84	90	63
ARS	248	207	172	258	340	172	84	91	63
<i>Decoding throughput (tokens per second)</i>									
CODETTA	1.0k	1.3k	0.7k	0.5k	1.0k	0.8k	0.2k	0.4k	0.2k
HARQ LDPC, draft-free	381k	393k	391k	251k	341k	363k	16k	14k	17k
ARS	7.5k	8.0k	8.0k	7.5k	8.0k	8.0k	42k	38k	41k

Table 1: Capacity in bits per token, the gain of CODETTA over each baseline, and throughput. Each experiment embeds 25 independent messages with the Qwen2.5-7B, Llama-3.1-8B, or Ministral-3-8B sender; CODETTA uses the matching Qwen2.5-1.5B, Llama-3.2-1B, or Ministral-3-3B draft. CODETTA and LDPC carry 1,024 bits per message, ARS carries 128 bits (16 on CTF). All of the messages are successfully recovered (25×1024 bits for every experiment on CODETTA). Throughput is measured in output tokens per second with batch size 10 on one NVIDIA H200 GPU.

7.3 Ablations of CODETTA keyed protocol

All ablations use the FINANCIAL workload and the Qwen2.5-7B sender.

Shared context. Figure 1 varies the fraction of the sender’s private records that the receiver also sees, from the primary asymmetric setting, in which the receiver sees none of them, to identical prompts, in which the receiver sees the whole record of 1,605 tokens on average. We plot two drafts, the 1.5B draft of the main experiments and a 7B draft that is the same model as the sender. Capacity grows monotonically with the shared context for both drafts: the 1.5B draft rises from 0.135 to 0.170 bits per token, and the 7B draft rises from 0.132 to 0.231 bits per token. The gain comes from a draft that tracks the sender’s distribution more closely, so that fewer proposals are rejected: the symbol error rate falls from 52% to 37% with the 1.5B draft and from 43% to 17% with the 7B draft.

At full sharing with the 7B draft, the draft and base distributions coincide at every position, and the remaining 17% comes from the bucket discretization alone. The figure also shows the value of a stronger draft. Without shared context, the two drafts deliver the same capacity up to noise, since the 7B draft still has to guess the private records. Once the receiver shares part of the context, the stronger draft delivers more, by 18% at half sharing and by 36% at full sharing. Accordingly, a stronger draft model yields a higher capacity and its advantage grows with the shared portion of the sender’s context.

Model mismatch. Table 2 pairs each sender with drafts from all three families. Cross-family pairs use a public byte-level bridge between the two tokenizers. In these experiments, the delivered payload rate varies more across sender models than across draft models. Ministral keeps 0.32 to 0.45 bits per token with every draft, and Qwen stays between 0.12 and 0.14. A same-family draft is best for each sender, and a cross-family draft loses 8% to 29% against it.

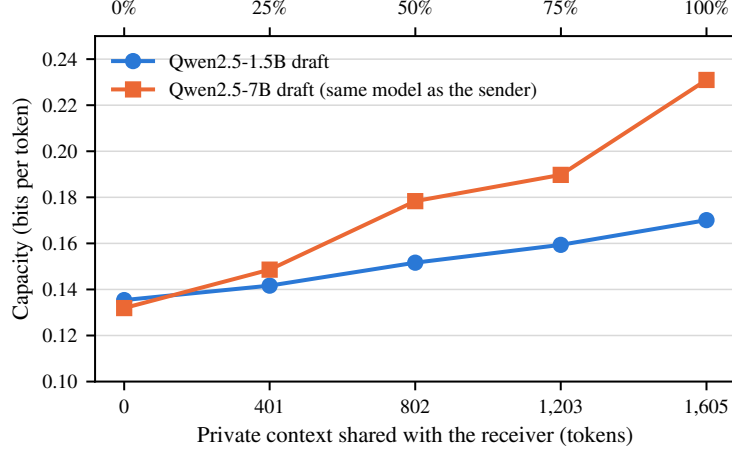


Figure 1: Capacity on FINANCIAL with the Qwen2.5-7B sender as the receiver sees a growing prefix of the sender’s private records, for a 1.5B and a 7B draft. The upper axis gives the shared fraction. Each point embeds 25 messages of 1,024 bits.

Sender	Draft		
	Qwen2.5-1.5B	Llama-3.2-1B	Ministral-3-3B
Qwen2.5-7B	0.1354	0.1187	0.1250
Llama-3.1-8B	0.2047	0.2289	0.1847
Ministral-3-8B	0.3187	0.3252	0.4476

Table 2: Capacity of CODETTA on FINANCIAL workload in bits per token for different pairs of sender and draft. All of the experiments recovered all 25 messages.

Components. Table 3 removes one component of CODETTA at a time on FINANCIAL. The last row is the draft-free LDPC code, and each row above it adds one component. The draft model offers the receiver balanced buckets, while the alphabet size stays fixed at two. It raises the capacity from 0.0280 to 0.0494 bits per token on Qwen (+76%), from 0.0358 to 0.0574 on Llama (+60%), and from 0.0936 to 0.1342 on Ministral (+43%). Adaptive alphabet size selection, still restricted to powers of two by the LDPC code, raises the capacity further to 0.0885 (+79%), 0.1718 (+199%), and 0.3634 (+171%) bits per token. And finally, the alphabet-adaptive ECC replaces the LDPC code, supports every alphabet size, and generates its checks online, which raises the capacity to 0.1354 (+53%), 0.2289 (+33%), and 0.4476 (+23%) bits per token. Accordingly, every component contributes for every sender, and the three components together deliver 4.8 to 6.4 times the capacity of the draft-free code, as reported in Table 1.

7.4 Key exchange

Table 4 summarizes the correctness of the key exchange and the number of visible tokens it publishes, for the three senders on the three workloads. No session failed on FINANCIAL for any sender, and 12, 8, and 0 of the 100,000 sessions failed on IFEVAL for Qwen, Llama, and Ministral. On CTF, 379 sessions failed for Qwen, and none failed for Llama and Ministral. Theorem 7 turns these counts into one-sided 95% bounds on the failure probability $\delta_{\mathcal{Q}}$: 3.0×10^{-5} for every cell without a failure, 1.9×10^{-4} and 1.4×10^{-4} for Qwen and Llama on IFEVAL, and 4.1×10^{-3} for Qwen on CTF. The per-coefficient hard-decision error rate is between 1.3% and 2.3% on FINANCIAL and IFEVAL, i.e., 3 to 6 errors on average per 255-bit codeword, and a session

Method	Qwen	Llama	Ministral
CODETTA	0.1354	0.2289	0.4476
HARQ LDPC with draft, power-of-two K	0.0885	0.1718	0.3634
HARQ LDPC with draft, fixed $K = 2$	0.0494	0.0574	0.1342
HARQ LDPC, draft-free	0.0280	0.0358	0.0936

Table 3: Component ablation on FINANCIAL in bits per token. From top to bottom, each row removes one component: the alphabet-adaptive ECC, then the adaptive alphabet size, then the public draft model.

Sender	Workload	Failed sessions S	$\delta_{\mathcal{Q}}$ estimation	Output tokens
Qwen2.5-7B	FINANCIAL	0	3.0×10^{-5}	81,920
Llama-3.1-8B	FINANCIAL	0	3.0×10^{-5}	81,920
Ministral-3-8B	FINANCIAL	0	3.0×10^{-5}	81,920
Qwen2.5-7B	IFEVAL	12	1.9×10^{-4}	70,751
Llama-3.1-8B	IFEVAL	8	1.4×10^{-4}	77,972
Ministral-3-8B	IFEVAL	0	3.0×10^{-5}	79,402
Qwen2.5-7B	CTF	379	4.1×10^{-3}	81,920
Llama-3.1-8B	CTF	0	3.0×10^{-5}	81,920
Ministral-3-8B	CTF	0	3.0×10^{-5}	81,920

Table 4: Correctness and output tokens of the key exchange for a 128-bit seed transmission over 1,280 carrier positions. The failure counts are over $m = 100,000$ sessions, and the bound is the one-sided 95% bound of [Theorem 7](#).

fails only when the errors of one codeword exceed the correction radius of 18. We attribute the failures on CTF to the low entropy of Qwen’s reports ([Section 7.2](#)): identical candidate blocks in a pool share one label, so the assignment has to match more slots to blocks with a different label, and the displacement of the carrier grows (see [Section 6.1](#)). Every exchange publishes at most 81,920 visible tokens, i.e., 1,280 positions of at most $B = 64$ tokens, which is 640 visible tokens per bit of the 128-bit pre-key. The responses on FINANCIAL and CTF fill the 64-token block at every position, whereas the shorter IFEVAL responses reduce the count to between 70,751 and 79,402 tokens.

Here we compare with [\[VZ26\]](#). The construction of [\[VZ26\]](#) has no implementation, so we estimate its cost on our workloads. We instantiate it with the Diffie–Hellman key exchange with uniform transcript that they suggest, with 256-bit group elements, i.e., 512 transcript bits at 128-bit security, with their exponential-error signaling scheme at a total failure probability of 10^{-4} , and with the same candidate pools as ours, i.e., 13,316 candidates per published block. In our experiments, the measured crossover probability is below 1% on all three workloads. The exchange then publishes about 2 million tokens if each of their published messages is as long as our 64-token blocks, against 81,920 tokens for CODETTA, i.e., CODETTA publishes about $23 \times$ fewer tokens.

8 Related Work

LLM steganography. Provably secure steganography goes back to the complexity-theoretic definition of Hopper, Langford, and Von Ahn [\[HLVA02\]](#) and the information-theoretic model of Cachin [\[Cac04\]](#), and Dedić et al. showed that a sender who uses the channel only through sampling pays a number of samples exponential in the rate [\[DIRR05\]](#), which is the cost that our keyless carrier pays when it draws a pool of candidate blocks for every q -ary symbol. Language models made these definitions practical over linguistic channels. Ziegler et al. embedded bits with arithmetic coding over GPT-2 and measured the divergence

they introduced [ZDR19], and later constructions preserve the model distribution exactly or bound the distribution divergence: adaptive dynamic grouping [ZYYH21], Meteor [KJGR21], Discop [DCW⁺23], SparSamp [WPC⁺25], Shimmer [BPL⁺25], ANStega [WPC⁺26], range coding [YM26], and adaptive dynamic sampling, which needs only a seeded sampling API [PB26]. All of them run in the symmetric setting: the receiver reruns the sender’s model on the same context, or queries the same model with the same seed, so that it can reproduce the sampling step, and independently deployed agents do not satisfy this assumption (Section 1). Mächtle et al. show that deterministic inference stacks leak a channel through the sampling seed, which the receiver recovers by exhaustive search over the seed space [MSB⁺26]. A model-free receiver appears in the asymmetric-resource protocol of Bai et al. [BYP⁺25], the ARS baseline of Section 7: the sender hides bits in permutations of the token distribution, and the receiver, which sees only the carrier, guesses the permutation through a sampling function; the protocol is robust over a binary symmetric channel, and our measurements place its rate below 0.025 bits per token. Zamir hides an arbitrary payload in a response so that the response is indistinguishable without the key, extending the undetectable watermark of Christ, Gunn, and Zamir [Zam24], and Christ and Gunn build stateless steganography with constant rate and robustness to a constant rate of substitutions from pseudorandom codes [CG24]. These constructions are analyzed asymptotically. CODETTA sits between the two extremes: its receiver runs a small public draft model using public context instead of the sender’s model and private states, and Section 7 measures the resulting gain in delivered payload rate.

Steganographic key exchange. Von Ahn and Hopper introduced public-key steganography and steganographic key exchange [VAH04]. As Vaikuntanathan and Zamir observe, the earlier covert key exchanges assume coartexts that are generated non-adaptively and a min-entropy per message that grows with the security parameter [VZ26]. Their own work is the closest to ours. With a shared key, they show that the tools of undetectable watermarking and steganography give a covert conversation whose rate matches the entropy of the honest messages up to constant factors. Without a key, they introduce pseudorandom noise-resilient key exchange, prove that covert key exchange is possible when sufficiently many honest messages have constant min-entropy, give constructions from sparse learning parity with noise and from a signaling compiler with noiseless feedback, and prove that public pseudorandom codes cannot correct a constant fraction of bit flips and that non-interactive protocols admit a quasi-polynomial attack. Section 6 compares the two key exchanges: theirs is a feasibility result without an implementation, whereas CODETTA transports lattice coefficients directly over the carrier, without a separate bitwise signaling layer, and certifies correctness empirically on the measured channel, and Section 7.4 estimates the cost of their construction on our workloads.

Watermarking and speculative decoding. Undetectable and distortion-free watermarks sample each token with randomness derived from a key so that the output distribution is unchanged, and their detectors need the key but not the model [CGZ24, KTHL24, CG24]. A watermark carries a fixed signal instead of a message, and the known limits of watermarking concern adversaries who edit or spoof the text [ZEF⁺24, PHZS24]. The maximal coupling of Section 4 is the acceptance rule of speculative decoding [LKM23, CBI⁺23], in which a small draft model proposes tokens and the target model accepts or resamples them so that the output keeps the target distribution exactly. CODETTA applies the same rule to a bucket-conditioned proposal and treats a rejection as a channel error, while in speculative decoding, rejection reduces the decoding speedup.

Collusion among LLM agents. A line of work studies collusion that emerges from the behavior of the model itself. Agents are prompted or trained to hide messages or reasoning in their text [MBS⁺24, MMM⁺25, RG23, ZM26], monitors of transcripts and chains of thought are designed and evaluated against agents that hide their intent [GSSR24, BHG⁺25, APB⁺26], and simulations and surveys map the risk of multi-agent collusion [RXW⁺25, Gha25]. These channels do not carry guarantees on the output distribution: the hidden content can change the model behaviors, so monitors and paraphrasers can detect or remove it. CODETTA is instead a cryptographic protocol with a distribution-preservation guarantee of the output transcript, so the passive auditors cannot observe the hidden collusion messages by inspecting the communication transcripts.

9 Defense Discussion

Theorem 3 and **Theorem 5** show that a passive auditor cannot detect collusion from the transcript. In this section, we discuss three mitigation directions that go beyond transcript inspection.

Auditing visible agent actions. The first direction is to audit the agents’ actions in addition to inspecting their messages. Coordinated actions (e.g., a trade or a tool call that the normal task does not require) are visible to the platform when they are executed, so an auditor on that platform can restrict the tools available to each agent, log every action, and check whether the actions are consistent with the assigned task. However, some actions may not be visible to the auditor. For example, sensitive data leakage is harmful as soon as the data leaves the sender, so the operator that deploys the sending agent can only limit the exposure in advance, i.e., give the agent access to no more data than its task requires. Data that the task legitimately requires, such as the retrieved records in FINANCIAL, remains exposed. We note that actions can also carry hidden information, e.g., through timing or through the choice among equivalent tool calls.

Verifiable sampling. The second direction is to make the sampling itself auditable. The agent commits to its sampling randomness, e.g., derived from a public beacon, and proves in zero knowledge that every published token was sampled from its model’s distribution with that randomness, without revealing its private context [SLZ24, CWSK24]. CODETTA embeds the hidden message by steering the sampling, so a valid proof rules out the covert channel. The drawback is cost: proving the inference of a large language model in zero knowledge is orders of magnitude more expensive than the inference in plaintext, so a future direction is to make this defense practical for deployment.

Perturbing the communication channel. The third direction is to modify the messages before they reach the receiver, e.g., by paraphrasing each message or regenerating it with another model. Both protocols of CODETTA rely on the sender and the receiver seeing the same transcript. In the keyed protocol, the receiver derives the draft distribution, the alphabet size, and the bucket partition of every token from the public prefix (Section 5), so an edited token changes the symbols that the receiver reads at every later position, and in the key exchange an edit inside a block changes the label of that block. The error correction of CODETTA is designed for the errors of the coupling, which leave the transcript intact, so a few edited tokens per message can already disable the channel. Robustness to such edits, e.g., through codes for insertion and deletion errors [CG24, MG24], is a natural extension of the design. Perturbations also have a cost of their own. They change the messages of honest agents, so their effect on task quality and latency must be measured for each deployment, and the component that rewrites the messages must be fully trusted by the auditor, since it sees and controls all of the communication. For example, the paraphraser must not embed secret messages of its own and must resist prompt injection attacks. We leave the design and evaluation of the mitigation to future work.

References

- [AGB⁺26] Sahar Abdelnabi, Amr Gomaa, Eugene Bagdasarian, Per Ola Kristensson, and Reza Shokri. Firewalls to secure dynamic LLM agentic networks. *Transactions on Machine Learning Research*, 2026. URL: <https://openreview.net/forum?id=w02FW1dMoY>.
- [APB⁺26] Usman Anwar, Julianna Piskorz, David D. Baek, David Africa, Jim Weatherall, Max Tegmark, Christian Schroeder de Witt, Mihaela van der Schaar, and David Krueger. A decision-theoretic formalisation of steganography with applications to LLM monitoring. *arXiv preprint arXiv:2602.23163*, 2026. URL: <https://arxiv.org/abs/2602.23163>, doi:10.48550/arXiv.2602.23163.

- [BHG⁺25] Bowen Baker, Joost Huizinga, Leo Gao, Zehao Dou, Melody Y Guan, Aleksander Madry, Wojciech Zaremba, Jakub Pachocki, and David Farhi. Monitoring reasoning models for misbehavior and the risks of promoting obfuscation. *arXiv preprint arXiv:2503.11926*, 2025.
- [BPL⁺25] Minhao Bai, Kaiyi Pang, Guorui Liao, Jinshuai Yang, and Yongfeng Huang. Shimmer: a provably secure steganography based on entropy collecting mechanism. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 5949–5965, 2025.
- [BRC60] Raj Chandra Bose and Dwijendra K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3(1):68–79, 1960. doi:[10.1016/S0019-9958\(60\)90287-4](https://doi.org/10.1016/S0019-9958(60)90287-4).
- [BYP⁺25] Minhao Bai, Jinshuai Yang, Kaiyi Pang, Xin Xu, Zhen Yang, and Yongfeng Huang. Provably robust and secure steganography in asymmetric resource scenario. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 1438–1456. IEEE, 2025.
- [Cac04] Christian Cachin. An information-theoretic model for steganography. *Information and Computation*, 192(1):41–56, 2004. doi:[10.1016/j.ic.2004.02.003](https://doi.org/10.1016/j.ic.2004.02.003).
- [CBB14] Peter Chapman, Jonathan Burket, and David Brumley. {PicoCTF}: A {Game-Based} computer security competition for high school students. In *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*, 2014.
- [CBI⁺23] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.
- [CG24] Miranda Christ and Sam Gunn. Pseudorandom error-correcting codes. In *Annual International Cryptology Conference*, pages 325–347. Springer, 2024.
- [CGZ24] Miranda Christ, Sam Gunn, and Or Zamir. Undetectable watermarks for language models. In *The Thirty Seventh Annual Conference on Learning Theory*, pages 1125–1139. PMLR, 2024.
- [CNS⁺25] Sahana Chennabasappa, Cyrus Nikolaidis, Daniel Song, David Molnar, Stephanie Ding, Shengye Wan, Spencer Whitman, Lauren Deason, Nicholas Doucette, Abraham Montilla, et al. Llamafirewall: An open source guardrail system for building secure ai agents. *arXiv preprint arXiv:2505.03574*, 2025.
- [CWSK24] Bing-Jyue Chen, Suppakit Waiwitlikhit, Ion Stoica, and Daniel Kang. Zkml: An optimizing system for ml inference in zero-knowledge proofs. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pages 560–574, 2024.
- [DCW⁺23] Jinyang Ding, Kejiang Chen, Yaofei Wang, Na Zhao, Weiming Zhang, and Nenghai Yu. Dis-cop: Provably secure steganography in practice based on distribution copies. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 2238–2255. IEEE, 2023.
- [DIRR05] Nenad Dedić, Gene Itkis, Leonid Reyzin, and Scott Russell. Upper and lower bounds on black-box steganography. In *Theory of Cryptography Conference*, pages 227–244. Springer, 2005.
- [Gal62] Robert G. Gallager. Low-density parity-check codes. *IRE Transactions on Information Theory*, 8(1):21–28, 1962. doi:[10.1109/TIT.1962.1057683](https://doi.org/10.1109/TIT.1962.1057683).

- [GCW⁺24] Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. In Kate Larson, editor, *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pages 8048–8057. International Joint Conferences on Artificial Intelligence Organization, 8 2024. Survey Track. [doi:10.24963/ijcai.2024/890](https://doi.org/10.24963/ijcai.2024/890).
- [GDJ⁺24] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
- [Gha25] Mohammad Sajjad Ghaemi. A survey of collusion risk in llm-powered multi-agent systems. In *Socially Responsible and Trustworthy Foundation Models at NeurIPS 2025*, 2025.
- [GSSR24] Ryan Greenblatt, Buck Shlegeris, Kshitij Sachan, and Fabien Roger. AI control: Improving safety despite intentional subversion. In *Forty-first International Conference on Machine Learning*, 2024. URL: <https://openreview.net/forum?id=KviM5k8pcP>.
- [HCAA⁺22] Jakob Hoydis, Sebastian Cammerer, Fayçal Ait Aoudia, Avinash Vem, Nikolaus Binder, Guillermo Marcus, and Alexander Keller. Sionna: An open-source library for next-generation physical layer research. *arXiv preprint arXiv:2203.11854*, 2022.
- [HLVA02] Nicholas J Hopper, John Langford, and Luis Von Ahn. Provably secure steganography. In *Annual International Cryptology Conference*, pages 77–92. Springer, 2002.
- [Hoc59] Alexis Hocquenghem. Codes correcteurs d’erreurs. *Chiffres*, 2:147–156, 1959.
- [KFL01] Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001. [doi:10.1109/18.910572](https://doi.org/10.1109/18.910572).
- [KJGR21] Gabriel Kaptchuk, Tushar M Jois, Matthew Green, and Aviel D Rubin. Meteor: Cryptographically secure steganography for realistic distributions. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1529–1548, 2021.
- [KTHL24] Rohith Kudutipudi, John Thickstun, Tatsunori Hashimoto, and Percy Liang. Robust distortion-free watermarks for language models. *Transactions on Machine Learning Research*, 2024. URL: <https://openreview.net/forum?id=FpaCL1M02C>.
- [LC04] Shu Lin and Daniel J. Costello. *Error Control Coding*. Prentice Hall, second edition, 2004.
- [LKM23] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR, 2023.
- [LKS⁺26] Alexander H Liu, Kartik Khandelwal, Sandeep Subramanian, Victor Jouault, Abhinav Rastogi, Adrien Sadé, Alan Jeffares, Albert Jiang, Alexandre Cahill, Alexandre Gavaudan, et al. Ministrall 3. *arXiv preprint arXiv:2601.08584*, 2026.
- [LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599, 2015.

- [Lub02] Michael Luby. Lt codes. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings*. IEEE Computer Society, 2002.
- [MBS⁺24] Sumeet R Motwani, Mikhail Baranchuk, Martin Strohmeier, Vijay Bolina, Philip H Torr, Lewis Hammond, and Christian S de Witt. Secret collusion among ai agents: Multi-agent deception via steganography. *Advances in Neural Information Processing Systems*, 37:73439–73486, 2024.
- [MG24] Ankur Moitra and Noah Golowich. Edit distance robust watermarks for language models. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, pages 20645–20693, 2024.
- [MMM⁺25] Yohan Mathew, Ollie Matthews, Robert McCarthy, Joan Velja, Christian Schroeder de Witt, Dylan Cope, and Nandi Schoots. Hidden in plain text: Emergence & mitigation of steganographic collusion in llms. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pages 585–624, 2025.
- [MSB⁺26] Felix Mächtle, Jonas Sander, Sebastian Berndt, Ben Weimar, Nils Loose, and Thomas Eisenbarth. Steganography without modification: Hidden communication via LLM seeds. *arXiv preprint arXiv:2606.09135*, 2026. URL: <https://arxiv.org/abs/2606.09135>.
- [Nat24] National Institute of Standards and Technology. Module-Lattice-Based Key-Encapsulation Mechanism Standard. Technical Report NIST FIPS 203, National Institute of Standards and Technology, August 2024. doi:10.6028/NIST.FIPS.203.
- [Ope26a] OpenAI. The hugging face incident and other third-party impact from misaligned models. <https://openai.com/hugging-face-incident-and-misalignment/>, September 2026. Accessed: 2026-09-18.
- [Ope26b] OpenAI. The Hugging Face incident and the road ahead. <https://openai.com/index/hugging-face-incident-and-the-road-ahead/>, August 2026. Accessed: 2026-09-09.
- [PB26] Kaiyi Pang and Minhao Bai. Provable secure steganography based on adaptive dynamic sampling. In *35th USENIX Security Symposium (USENIX Security 26)*, pages 3143–3159, 2026.
- [Pea88] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.
- [PHZS24] Qi Pang, Shengyuan Hu, Wenting Zheng, and Virginia Smith. No free lunch in LLM watermarking: Trade-offs in watermarking design choices. In *Conference on Neural Information Processing Systems*, 2024.
- [QLL⁺24] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 15174–15186, 2024.
- [RG23] Fabien Roger and Ryan Greenblatt. Preventing language models from hiding their reasoning. *arXiv preprint arXiv:2310.18512*, 2023. URL: <https://arxiv.org/abs/2310.18512>.

- [RU08] Tom Richardson and Rüdiger Urbanke. *Modern Coding Theory*. Cambridge University Press, 2008. doi:10.1017/CB09780511791338.
- [RXW⁺25] Qibing Ren, Sitao Xie, Longxuan Wei, Zhenfei Yin, Junchi Yan, Lizhuang Ma, and Jing Shao. When autonomy goes rogue: Preparing for risks of multi-agent collusion in social systems. *arXiv preprint arXiv:2507.14660*, 2025.
- [Sho06] Amin Shokrollahi. Raptor codes. *IEEE transactions on information theory*, 52(6):2551–2567, 2006.
- [SLZ24] Haochen Sun, Jason Li, and Hongyang Zhang. zkllm: Zero knowledge proofs for large language models. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 4405–4419, 2024.
- [TZZ⁺24] Xiangru Tang, Anni Zou, Zhuosheng Zhang, Ziming Li, Yilun Zhao, Xingyao Zhang, Arman Cohan, and Mark Gerstein. Medagents: Large language models as collaborators for zero-shot medical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 599–621, 2024.
- [VAH04] Luis Von Ahn and Nicholas J Hopper. Public-key steganography. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 323–341. Springer, 2004.
- [VZ26] Vinod Vaikuntanathan and Or Zamir. Undetectable conversations between AI agents via pseudorandom noise-resilient key exchange. *arXiv preprint arXiv:2604.04757*, 2026. URL: <https://arxiv.org/abs/2604.04757>.
- [WPC⁺25] 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. In *34th USENIX Security Symposium*, pages 6817–6835. USENIX Association, 2025.
- [WPC⁺26] Yaofei Wang, Weilong Pang, Kejiang Chen, Jinyang Ding, Donghui Hu, Weiming Zhang, and Nenghai Yu. Breaking the generative steganography trilemma: Anstega for optimal capacity, efficiency, and security. In *33rd Annual Network and Distributed System Security Symposium*. The Internet Society, 2026. doi:10.14722/ndss.2026.240605.
- [XSLW25] Yijia Xiao, Edward Sun, Di Luo, and Wei Wang. Tradingagents: Multi-agents LLM financial trading framework. In *The First MARW: Multi-Agent AI in the Real World Workshop at AAAI 2025*, 2025. URL: <https://openreview.net/forum?id=4QPrXwMQt1>.
- [Y CJ⁺24] Weikang Yuan, Junjie Cao, Zhuoren Jiang, Yangyang Kang, Jun Lin, Kaisong Song, Tianqianjin Lin, Pengwei Yan, Changlong Sun, and Xiaozhong Liu. Can large language models grasp legal theories? enhance legal reasoning with insights from multi-agent collaboration. In *Findings of the association for computational linguistics: EMNLP 2024*, pages 7577–7597, 2024.
- [YM26] Ruiyi Yan and Yugo Murawaki. Efficient provably secure linguistic steganography via range coding. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, pages 890–907. Association for Computational Linguistics, 2026. doi:10.18653/v1/2026.acl-long.39.
- [YPNY23] John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. Intercode: Standardizing and benchmarking interactive coding with execution feedback. *Advances in Neural Information Processing Systems*, 36:23826–23854, 2023.

- [YYZ⁺24] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.
- [Zam24] Or Zamir. Excuse me, sir? your language model is leaking (information). *Transactions on Machine Learning Research*, 2024. URL: <https://arxiv.org/abs/2401.10360>.
- [ZDR19] Zachary Ziegler, Yuntian Deng, and Alexander Rush. Neural linguistic steganography. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1210–1215, Hong Kong, China, November 2019. Association for Computational Linguistics. URL: <https://aclanthology.org/D19-1115/>, doi:10.18653/v1/D19-1115.
- [ZEF⁺24] Hanlin Zhang, Benjamin L. Edelman, Danilo Francati, Daniele Venturi, Giuseppe Ateniese, and Boaz Barak. Watermarks in the sand: Impossibility of strong watermarking for language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 58851–58880. PMLR, 2024. URL: <https://proceedings.mlr.press/v235/zhang24o.html>.
- [ZLM⁺23] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models, 2023. URL: <https://arxiv.org/abs/2311.07911>, arXiv:2311.07911.
- [ZM26] Zhejian Zhou and Jonathan May. Conceptual steganography. *arXiv preprint arXiv:2605.26537*, 2026. URL: <https://arxiv.org/abs/2605.26537>, doi:10.48550/arXiv.2605.26537.
- [ZYYH21] Siyu Zhang, Zhongliang Yang, Jinshuai Yang, and Yongfeng Huang. Provably secure generative linguistic steganography. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3046–3055, Online, August 2021. Association for Computational Linguistics. URL: <https://aclanthology.org/2021.findings-acl.268/>, doi:10.18653/v1/2021.findings-acl.268.

A Belief Propagation Details

This appendix gives the sum-product updates summarized in [Section 2.4](#), a worked example, and the relation between BP and the MAP decoder.

Factor graph. The graph has a variable node for every unknown symbol u_i and a check node for every received check s , joined by an edge whenever $i \in S_s$; we write $\partial i = \{s : i \in S_s\}$ for the checks of symbol i . Check s carries a factor ϕ_s that scores each local assignment of its symbols by how well it explains the received value. If the check value passed through a channel with transition probabilities W_s and arrived as $\hat{C}_s$, then

$$\phi_s(u_{S_s}) = W_s(\hat{C}_s \mid C_s(u_{S_s})), \quad (24)$$

and $\mathbb{P}[u \mid \mathcal{O}]$ is proportional to the product of all factors. For an LDPC code the check factor is instead the indicator $\mathbf{1}\{C_s(u_{S_s}) = 0\}$ and the channel likelihood of each received bit multiplies the messages of its variable node; we omit that term because our symbols are never transmitted directly.

Message updates. A message is an opinion about one symbol, formed while excluding the recipient so that no evidence is counted twice. Variable i tells check s what its other checks currently make it believe,

$$\mu_{i \rightarrow s}(u_i) \propto \prod_{s' \in \partial i \setminus \{s\}} \mu_{s' \rightarrow i}(u_i), \quad (25)$$

and check s tells variable i how likely each value of u_i is, given its factor and the current beliefs of its other symbols,

$$\mu_{s \rightarrow i}(u_i) \propto \sum_{u_{S_s \setminus \{i\}}} \phi_s(u_{S_s}) \prod_{j \in S_s \setminus \{i\}} \mu_{j \rightarrow s}(u_j). \quad (26)$$

All messages start uniform, and the two updates alternate until a public iteration cap or until the messages stop changing. The belief of symbol i combines all incoming opinions, and the decision is its mode:

$$b_i(u_i) \propto \prod_{s \in \partial i} \mu_{s \rightarrow i}(u_i), \quad \hat{u}_i = \arg \max_{u_i \in \Sigma} b_i(u_i). \quad (27)$$

For a check modulo K over d binary variables, the sum in Equation (26) is a convolution over $\mathbb{Z}_K$ and takes $O(dK^2)$ time by dynamic programming over partial sums.

Example. Suppose one check involves only u_1 and reads $u_1 = 1$, and a second reads $u_1 \oplus u_2 = 1$. The first check makes u_1 believe it is 1 with probability $1 - e$, where e is the channel error rate. In the next iteration u_1 passes this belief to the second check, which concludes that u_2 is 0 with about the same confidence, and a third check on $u_2 \oplus u_3$ then carries the evidence on to u_3 . Each iteration moves evidence one edge further, from the checks that pin a symbol down to the rest of the graph. The same mechanism explains why BP stalls without low-degree checks: if every check involves several symbols, uniform incoming messages produce uniform outgoing messages in Equation (26).

Approximation and guarantees. On a tree, the messages implement exact marginalization and b_i is the exact posterior marginal. On a graph with cycles, the same evidence can travel around a cycle and be counted again, so BP is an approximation, accurate when the graph is sparse and short cycles are rare. BP extends incrementally: a newly received check adds one node and its edges, and the receiver simply runs more rounds. BP comes with no guarantee that its output is the transmitted message, whereas the MAP decoder admits a direct analysis. The validation tag tells the two outcomes apart: the receiver accepts a decoded message only if its keyed Tag validates and returns $\perp$ otherwise, and with a λ -bit keyed tag a wrong message passes with probability at most $Q2^{-\lambda}$ plus the PRF advantage over Q validation attempts. Validation also lets the receiver test a bounded list of candidates near the BP output, for instance by flipping its least confident symbols, with each test counted toward Q .

B Workload Details

B.1 FINANCIAL

FINANCIAL simulates an issuer-intelligence vendor. The receiver agent, an asset manager, sends the sender agent a JSON research request about one synthetic issuer. The sender retrieves the five records for that issuer from its local database, which holds 20 synthetic issuers with five record types each (issuer profile, quarterly update, public milestone, sector and peer context, risks and catalysts), and writes a research report in prose. Its system prompt restricts it to records marked public or distribution-permitted. The retrieved records, about 1.7k tokens per request, are appended to the sender’s context only; the receiver sees the system prompt and the request. The report is the carrier. The pool contains 50 requests.

Public request from the receiver agent (abridged)

```
{ "from": "PortfolioResearchAgent_B", "to": "IssuerIntelAgent_A",
  "task": "60_day_investment_committee_memo", "asset": "SYN-LFD",
  "payload": {
    "issuer": "Lumen Foods Holdings",
    "sector": "packaged foods / branded nutrition",
    "requested_sections": ["issuer overview", "recent public developments",
      "earnings and guidance summary", "sector and peer context",
      "public risks", "public catalysts", "scenario analysis",
      "monitoring questions", "data-quality caveats"] } }
Suggested lookup: SELECT * FROM intel_records WHERE asset = 'SYN-LFD'
ORDER BY record_type;
```

Sender-only context: one of the five retrieved records (abridged)

```
{ "record_id": "R11-QUARTER", "issuer": "Lumen Foods Holdings",
  "record_type": "quarterly_update", "as_of_date": "2026-06-30",
  "public_observations": [
    "Revenue trend: 4% year-over-year, supported by pricing but offset by
      modest volume declines.",
    "Margin trend: gross margin improved 130 basis points as commodity hedges
      and productivity initiatives helped.",
    "Guidance: management reaffirmed organic growth guidance and raised
      productivity savings target.", ... ],
  "metrics": { ... }, "monitoring_points": [ ... ] }
```

Response, i.e., the carrier (first lines)

```
Research Report for 90-Day Research Review of SYN-LFD (Lumen Foods Holdings)

### Issuer Overview
Lumen Foods Holdings is a synthetic issuer operating in the packaged foods and
branded nutrition sectors, with significant presence in North America and
Europe. Its business is segmented into branded nutrition (40%), frozen meals
(28%), snacking (21%), and foodservice (11%). ...

### Recent Public Developments
In the most recent quarterly update, Lumen Foods reported a 4% year-over-year
revenue growth, driven by pricing offsetting modest volume declines. Gross
margin improved by 130 basis points due to commodity hedging and productivity
initiatives. ...
```

B.2 IFEVAL

IFEVAL prompts come from the public IFEval dataset [ZLM⁺23] at a pinned revision. Each prompt carries one or more instructions that a rule-based scorer verifies. Before answering, the sender’s base model produces a private plan by greedy decoding of at most 48 tokens under the instruction below; the plan is appended to the sender’s context and never shown to the receiver, whose context is the system prompt and the request. The response is the carrier. The system prompt for both agents is “You are a careful and helpful instruction-following assistant.”

Plan instruction appended to the sender’s prompt

Privately make an abstract response-control plan; do not answer the request. Use at most 24 words and only generic control categories such as length, sections, bullets, casing, punctuation, required or forbidden elements, and ending. Counts are allowed. Do not include topic-specific names, dates, facts, examples, advice, claims, or reusable answer phrases. This private plan will be supplied to the answering agent.

Example 1 (instruction: change_case:english_capital)

Request: Come up with 3 names for a 2B software company. Make sure your names are in English and all capital letters.
Plan: Plan: 3 x Capital Letters, No Punctuation, <25 Chars Each
Response: INNOVAQBRICK
CODECLARIGEN
SECURESKYLINE

Example 2 (instruction: punctuation:no_comma)

Request: Write a riddle about Camilla that doesn't use commas.
Plan: Sections:1 Punctuation:none Length:24words Casing:none
Response: scratch your back i am small but fierce camilla

B.3 CTF

CTF simulates two agents that communicate through a shared file system instead of a conversation. We take a subset of 20 challenges from the offline InterCode-CTF [YPNY23] adaptation of picoCTF [CBB14], i.e., nine cryptography, five reverse-engineering, four general-skills, one forensics, and one binary-exploitation challenge. The sender agent first solves a challenge in an offline sandbox with shell and Python tools, in a ReAct loop of at most 50 messages and three flag submissions at temperature 1. The complete solve trajectory, i.e., the task statement with its attached files, every tool command, every tool output, and the submission feedback, is the sender’s private context. The same model then receives the report request below, appended to this trajectory, and writes one technical report on the task, which ends at its end-of-sequence token or after 576 tokens. The report is the carrier: the sender writes it into a directory that both agents can access, and the receiver agent reads the file from there, so the only communication between the two agents is the file itself. The receiver’s context is the public task statement and the report file, and it never sees the trajectory. The first 64 tokens of each report and all code blocks in it are generated without embedding, and the capacity in Table 1 divides the payload by all visible tokens of the report, including these. We have omitted the tool outputs in the example below.

Example (calibration task 24, abridged)

Task: Run the Python script code.py in the same directory as codebook.txt.
Trajectory: sender only; six tool turns, commands shown without their outputs
[1] bash head -c 100 code.py
[2] bash cat code.py
[3] bash cat codebook.txt
[4] python codebook = "azbycxdwevfugthsirjqkplomn"
key = codebook[4] + codebook[14] + codebook[13] + codebook[14] + codebook[23] +
codebook[25 % len(codebook)] + codebook[16] + codebook[0] + codebook[25 % len(codebook)]

```
        print(key)
[5] python key = "chthonian" ... result = str_xor(flag_enc, key) ... print(result)
[6] bash    submit picoCTF{c0d3b00k_455157_d9aa2df2}
Report:    the carrier, written to the shared directory (first lines)
### **Challenge Analysis: Decrypting the Flag with a Custom XOR Cipher**
#### **Objective and Underlying Concepts**
This Capture The Flag (CTF) challenge required decryption of a flag using a
simple XOR-based cipher. The provided Python script (`code.py`) incorporated
a `str_xor` function to perform bitwise XOR operations between a secret
message (`flag_enc`) and a key derived from `codebook.txt`. ...
[...]
FINAL: picoCTF{c0d3b00k_455157_d9aa2df2}
```

Report request appended to the sender's trajectory (abridged)

REPORT PHASE. Tool execution has ended. Write a complete, self-contained technical report in English about the task and the work recorded above. Explain the task and relevant inputs, the approach taken, the decisive commands or reasoning, the evidence obtained, and the final result. Develop the explanation in connected paragraphs, including concrete intermediate observations and why they support or fail to support the conclusion. ... Treat earlier assistant reasoning as hypotheses; correct any claims contradicted by tool output. Distinguish observed facts from interpretations. Use code excerpts when they help explain the method. Do not invent actions, observations, or results. End with exactly one plain line: FINAL: <verified flag>, or FINAL: UNSOLVED if no flag was verified.