Optimal Error Correction for Computationally Bounded Noise*

Silvio Micali
MIT

Chris Peikert
SRI International†

Madhu Sudan
MIT

David A. Wilson
MIT

May 24, 2010

Abstract

For adversarial but *computationally bounded* models of error, we construct appealingly simple and efficient cryptographic encoding and *unique* decoding schemes whose error-correction capability is much greater than classically possible. In particular:

1. *For binary alphabets, we construct positive-rate coding schemes that are uniquely decodable under a $1/2 - \gamma$ error rate for any constant $\gamma > 0$.*
2. *For large alphabets, we construct coding schemes that are uniquely decodable under a $1 - R$ error rate for any information rate $R > 0$.*

Our results for large alphabets are actually optimal, since the "computationally bounded but adversarial channel" can simulate the behavior of the q-ary symmetric channel, where q denotes the size of the alphabet, the capacity of which is known to be upper-bounded by $1 - R$.

Our results hold under minimal assumptions on the communication infrastructure: namely, (1) we allow the channel to be more powerful than the receiver, and (2) we only assume that some information about the sender —a *public key*— is known. (In particular, we do not require any shared secret key or joint local state between sender and receivers.)

1 Introduction

Let us briefly recall the setting of the classical theory of error correction [Sha48, Ham50]:

> A *sender* starts with some *message*, which is represented as a string of symbols over some *alphabet*. The sender *encodes* the message into a longer string over the same alphabet, and transmits the block of data over a *channel*. The channel introduces *errors* (or *noise*) by changing some of the symbols of the transmitted block, then delivers the corrupted block to the *receiver*. Finally, the receiver attempts to *decodes* the block, hopefully to the intended message. Whenever the sender wants to transmit a new message, the process is repeated.

Classically, two quantities are of special interest in this setting. The first is the *information rate*, the ratio of the message length to the encoded block length. The second is the *error rate*, the ratio of the number of errors to the block length. Of course, we desire coding schemes that tolerate high error rates while simultaneously having large information rates. (In practice, smaller alphabets are desirable too.)

The heart of our work is to prove that the achievable limits for these and other quantities crucially depends on the assumed nature of the noise introduced by the communication channel.

*A preliminary version of this paper appeared in the proceedings of the 2005 Theory of Cryptography Conference.

†This work was performed while at MIT.

1.1 Modeling Noisy Channels

There are two historically popular ways to model a noisy channel for communication. In his pioneering 1948 work, Shannon considered a *probabilistic channel*, which perturbs each transmitted symbol independently with some fixed probability. Shannon demonstrated how information can be encoded to withstand such noise with probability arbitrarily close to 1/2 for binary channels, and arbitrarily close to 1 for channels with large alphabets. One could argue, however, that Shannon's notion of a probabilistic channel may be too simplistic for modeling actual sources of noise. In the real world, data corruption can be the result of complex interactions among many factors both intrinsic and extrinsic to the communication system, and errors tend to be distributed neither uniformly nor independently. For example, both radio interference and physical defects on storage media tend to be "bursty," with periods of very high noise separated by spans of quiescence.

By contrast, Hamming effectively proposed an *adversarial channel* that perturbs symbols in a *worst-case* fashion subject only to an upper bound on the number of errors per block of data. Hamming's model is of course more general and thus "safer" than Shannon's one. This generality, however, comes at a price: it imposes severe limits on the performance of codes. For instance, in order to correctly decode the intended message under a binary alphabet, the error rate of the channel must not exceed $1/4$ (unless the information rate is 0).

Computationally Bounded Adversarial Channels. As described above, "adversarial channels are *computationally unbounded*." That is, in principle, the channel may employ unbounded amount of computation to decide *where* and *how* to corrupt the transmission. This, however, is an overly pessimistic, and possibly outright unrealistic assumption. Natural processes, included those causing errors in communication channels, are carried out by Nature in a feasible manner. From a computer science point of view, this implies that even the most adversarial channel can only use a feasible amount of computing to decide where and how to cause transmission errors. Indeed, in 1994, Lipton proposed the notion of a *feasible* channel, which is essentially a Hamming channel restricted to *polynomial-time* computation [Lip94]. However, for over a decade after this proposal, both a rigorous formalization of computationally bounded channels and error-correcting schemes for such channels have been conspicuously absent. In this paper we aim to remedy both problems.

The Importance of State. We stress that modeling the channel as an adversarial computational process introduces concerns not arising in either the simple model of a probabilistic channel or the computationally unbounded adversarial channel. In particular, central to every computational model (be it a Turing machine or a modern computer) is the notion of *state*, that is, what a computational process is allowed or required to know beforehand and to retain from one computation to the next.

In the case of a computationally bounded adversarial channel, it does matter whether the channel is given knowledge of the identity of the sender and the receiver, and the details of the encoding scheme. And even more it matters whether the adversarial channel can retain memory of *past* transmissions. Note that the issue of "state" is totally absent in the simplest case of a probabilistic channel, since by definition "each error is independent." More sophisticated models of probabilistic channels allow for some Markovian behavior among errors, and thus for some state retention. However, it is customary to assume that the length of the transmission is unbounded while the state information is finite, thereby minimizing any serious influence of state from one transmission to the next. For the adversarial, computationally unbounded, channel the issue of state is also irrelevant, since the adversary is by definition free to choose the worst error possible for every transmission. By contrast, retaining state does in principle enable a computationally bounded adversarial

channel to use the transmission history in order to *monitor* how well it is doing, and *learn* how inject errors that are harder or impossible to recover from.

The ability to maintain state is also very relevant for the sender and the receiver, as they too ultimately are computational processes. And so is any a priori knowledge that the sender and the receiver may have (e.g., about each other's identities). Of course, the more the channel is allowed to know and the less senders and receivers are required to know the better the model and the better the results.

1.2 Our Contributions

Our Setting. Our precise model is detailed in Section 3. Informally, sender, recipient and channel are (possibly probabilistic) polynomial-time algorithms capable of maintaining state.

The sender has a public key pk and a matching secret key sk, and needs to keep at least a loose track of the number of messages sent. (He needs to use a counter —e.g., a clock— whose content increases with the number of messages sent.) To encode a message, he uses sk and the current content of his counter. In practice, 100 bits of state are sufficient to keep track of the counter, and a few hundred bits suffice for sk. Once he already has selected his public and secret key, the sender can be deterministic.[1]

The receiver can always be deterministic and needs to know even less: namely, pk, "the cryptographic identity" of the sender. He decodes each message based only on the received string and pk. (In particular, the decoding algorithm is always the same.)

The channel can be probabilistic, if it so wants. It has an unbounded state, and can use it to record essentially anything known to the receiver and/or the sender, except for sk. In particular, at the beginning of each transmission, the state of the channel is allowed to contain the sender's pk, the exact number of messages transmitted so far, the exact sequence of the messages encoded (including the current one), the exact sequence of their encodings (including the current one), the exact sequence of all errors it introduced in the past, and the exact sequence of the receiver's past decodings (correct or incorrect as they may be).[2] Based on all such inputs, the channel it allowed to compute the errors he wants to introduce.

Notice that we do not explicitly bound the number of errors that the adversarial channel introduces. (Indeed, it is free to introduce 99% or even more errors.) However, as we shall see, *for any given transmission* our solution guarantees a correct and unique decoding whenever the number of errors introduced in *that transmission* is below the prescribed limit.

Our results. In our setting, we construct coding schemes both for binary and large alphabets; namely:

1. *For the binary alphabet, we construct positive information rate coding schemes that are uniquely decodable under error rate* $1/2 - \gamma$*, for any constant* $\gamma > 0$*.*

2. *For large alphabets, we construct positive information rate coding schemes that are uniquely decodable under error rate* $1 - \gamma$*, for any constant* $\gamma > 0$*.*

Recall that the seminal paper of Hamming [Ham50] points out that a code of relative distance δ can correct at most a fraction $\delta/2$ of adversarial (in our language) errors. Combining Hamming's with Plotkin's result

[1] Note that probabilism is required for choosing any secret key, else "any one could predict the secret by running the secret-selection algorithm." But after sk has been chosen, the only advantage for the sender being probabilistic may be computational efficiency.

[2] Indeed, as the receiver's decoding algorithm is public and solely based on pk and the received string (both known to the channel), the channel can always predict what the decoding is for the errors it considers.

[Plo60]—asserting that a binary code of positive rate can have relative distance of at most 1/2— we get that a $1/4 - \gamma$ error rate is the best possible for unique decoding (and positive information rate) over an adversarial and computationally unbounded channel over the binary alphabet. Thus by making the channel reasonable we are able to bypass the classical barrier on the error rate.

Recall too that it is impossible to decode above an error rate of 1/2 even for Shannon's simple probabilistic channel model. Therefore our result allows us to handle a very adversarial channel with the same error rate achievable for a very benign channel.

Similarly, for large alphabets, combining Hamming's result with the trivial observation that no code can have relative distance greater than 1, we get that $1/2 - \gamma$ is the maximal error rate for unique decoding over an adversarial and computationally unbounded channel. Again therefore we are able to bypass this classical bound and are able to handle quite adversarial channels with arbitrarily high error rate.

Simplicity, Modularity, and Automatic Improvements. Our results are obtained by a very simple and modular combination of two powerful components, namely, *digital signature* and *list-decoding*. In essence, our scheme implements an age-old practice of signing and dating messages! Specifically, the sender time-stamps the message by adding the contents of the current counter, *digitally* signs the time-stamped message, and then encodes the result using a *list-decodable* code. The receiver, list-decodes the received string, and then outputs as the *unique* message the element of the list that is properly signed and has the largest time-stamp.

An immediate consequence of this modular approach is that any improvement to the performance of either component yields a corresponding improvement in the performance of our coding scheme. Indeed, using the recent advances in list-decoding due to Parvaresh and Vardy [PV05], and Guruswami and Rudra [GR08] (obtained after the original version of this result [MPSW05]), our above-stated result for large alphabets automatically improves as follows:

2′. *For large alphabets, we construct coding schemes that are uniquely decodable under error rate* $1 - R - \gamma$*, with information-rate* R*, for any constants* $R, \gamma > 0$.

Recall that for an error rate of $1 - R$, the Shannon capacity of the q-ary symmetric channel is at most R for every q. Note that the computationally bounded but adversarial channel can simulate the actions of the q-ary symmetric channel. This is so because an adversarial channel can simply choose to just flip each transmitted character with a fixed probability independently for each character, a behavior that only requires linear computation time (and no memory at all). Thus, the capacity of the computationally bounded adversarial channel is bounded from above by that of the q-ary symmetric channel, and thus by R.

Thus our result achieves the optimal tradeoff between information rate and error rate for sufficiently large alphabets. Thus, again, we are able to handle quite adversarial channels with the same performance as very benign channels (now for every information rate).

1.3 Related Work

We wish to contrast our results with other lines of work which attempt to go beyond the Hamming bound for error correction.

Secret and Shared Randomness. In [Lip94], Lipton considered going beyond the Hamming bound for adversarial but computationally bounded channels. This works considers the transmission of a single message m, assuming that the sender and the receiver share beforehand a random string ρ (much shorter than m

under proper cryptographic assumptions). The result crucially relies on the fact that ρ is *secret* that is, totally unknown to the channel. Under the same cryptographic assumptions, Ding, Gopalan, and Lipton [DGL06] later extended this work to the transmission of multiple messages with a fixed, short secret ρ.

The fact that senders and receivers share a common secret, makes these works not suitable to many settings, such as message broadcast. By contrast, notice that our results are suitable for broadcasting messages, since the sender need not share any secret with any of its potential recipients, nor does he need to know their identities. In addition, the scheme of [DGL06] assumes that sender and receiver maintain properly *synchronized states*, a restriction absent in our result.

For the transmission of a single message, Langberg [Lan04] has succeeded in extending the Lipton result to computationally unbounded channels, but at the price of making recipient computationally unbounded as well.

List Decoding. List decoding, as defined by Elias and Wozencraft [Eli57, Woz58], has been used recently by Sudan [Sud97] and Guruswami and Sudan [GS99] to go beyond the Hamming bound. However, we stress that in their setting the decoder is allowed to output a list of candidate messages (guaranteed to include the correct message), while in our setting we insist that the decoder outputs the unique, correct message.

However, our result shows that list decoding can be used to yield unique decodability for all practical purposes.

Side Channels. Guruswami [Gur03] shows how to bypass the Hamming bound for messages transmitted over an adversarial channel, even if computationally unbounded, provided that sender and receiver are able to communicate via an *additional* and *noiseless* side channel. (The interest of this result comes from the fact that the side channel is only "sparingly" used.)

By contrast, our sender and receiver are not assumed to have any side channels for their communication.

2 Preliminaries

Our solution depends on two main notions: (1) list-decodable codes —from coding theory— and (2) digital signatures —from cryptography. We thus introduce notation and briefly recall these two notions.

2.1 Notation

For a positive integer n, we write $[n] = \{1, \ldots, n\}$.

For a finite set Σ and a nonnegative integer n, we will refer to an element of Σ^n as a *string* or *word* over the *alphabet* Σ. For a word $x \in \Sigma^n$, we write x_i to denote the ith component of x.

Asymptotics. For positive functions $f(n), g(n)$, we write $f(n) = O(g(n))$ and/or $g(n) = \Omega(f(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = C$ for some absolute constant $C \geq 0$. We write $f(n) = o(g(n))$ and/or $g(n) = \omega(f(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

We say that a function $f(n)$ is *negligible* in n if, for every positive polynomial $p(n)$, $f(n) = o(1/p(n))$.

2.2 Coding Theory Background

For a finite set Σ and positive real $R < 1$, an *error-correcting code* over an *alphabet* Σ and *information rate* R is the image of a mapping $C : \Sigma^k \to \Sigma^n$ where $n = n(k) = \lfloor k/R \rfloor$.

We refer to k as the *message length*, to n as the *block length*, to C as the encoding function, and to the elements of the set $\left\{C(x) \ : \ x \in \Sigma^k\right\}$ as *codewords*. We will often overload notation and write C to refer to the set of codewords (as well as the mapping).

The *Hamming distance* between two words $x, y \in \Sigma^n$ is the number of positions i in which x_i and y_i differ: $\Delta(x, y) = |\{i \in [n] \ : \ x_i \neq y_i\}|$. It is easy to verify that the Hamming distance is a metric. A code C has *distance* (at least) d if $\Delta(C(x), C(y)) \geq d$ for every distinct $x, y \in \Sigma^k$.

We also consider asymptotics of infinite *families* of codes of information rate R:[3]

$$\mathbb{C} = \left\{C_k \ : \ \Sigma^k \to \Sigma^n\right\}_{k \in \mathbb{N}}.$$

A family has *relative distance* δ if for every k, C_k has distance $\delta \cdot n$. The aim of most error-correcting codes is to have R and δ as high as possible.

Computational complexity and List Decodability. Let $\mathbb{C} = \{C_k\}$ be a family of codes.

We say that $\mathbb{C}$ is *explicitly constructible* if (1) it has a single efficient (deterministic polynomial-time) algorithm E for computing all the encoding functions C_k, that is $E(k, \cdot) = C_k(\cdot)$.

We say that $\mathbb{C}$ is (ρ, L)*-list decodable* if, for every k and $r \in \Sigma^n$, there are at most $L(k)$ codewords within distance $\rho \cdot n$ of r. We further say that $\mathbb{C}$ is (ρ, L) *efficiently list decodable* if it has a single polynomial-time *list decoding* algorithm $\mathcal{LD}$ which, on input a string $r \in \Sigma^n$, outputs all the codewords in C_k within distance $\rho \cdot n$ of r.

Accordingly, an efficiently list decodable code $\mathbb{C}$ is a pair: $\mathbb{C} = (E, \mathcal{LD})$.

2.3 Digital Signatures

The intuitive notion of a digital signatures was initially proposed by Diffie and Hellman [DH76], but not fully formalized and rigorously constructed until Goldwasser, Micali, and Rivest [GMR88]. In essence, a signer generates a pair of matching keys (strings), (pk, sk). We refer to pk as the verification key and to sk as the signing key. The signer uses sk to sign a message m, that is, to compute a short string σ called the signature of m. Given pk, anyone can easily verify that σ is the signature of m. But no one should, without knowing sk, be able to efficiently compute the signature of a message, even with knowledge of pk. Thus it is in the interest of the signer to publicize pk as his verification key, and to keep sk secret. By so doing, he enable all users to verify his own signatures, while retaining for himself the power of signing messages. For this reason, pk is mostly referred to as the *public key* and sk as the *secret key.* Let us now elaborate on the syntax and the semantics of a signature scheme, that is on the scheme's algorithmic ingredients and security properties.

Syntactically, a digital signature scheme $\mathbb{S}$ consists of a triple of polynomial-time deterministic algorithms (G, S, V) with the following functionality.

1. Algorithm G is the key generator. On input a random binary string r, G outputs a pair of matching keys (pk, sk), each having the same length as r. We refer to r's length as the security parameter.

2. Algorithm S is the signing algorithm. On input two binary strings sk and m, S outputs a binary signature σ.

[3] Some of the families in the literature, such as Reed-Solomon codes, use a more general definition, allowing the alphabet to depend on k. While our methods easily work for such more general notions, for simplicity we are happy to state all of our results for fixed-alphabet codes.

3. Algorithm V is the verification algorithm. On input three binary strings, pk, m and σ, V outputs either YES or NO with the following basic constraint: if, for some r, $G(r) = (pk, sk)$, and if $\sigma = S(sk, m)$, then $V(pk, m, \sigma) =$YES.

Semantically, we must require that an adversary should not be able to forge signatures. At the lowest level, this may be taken to mean that an attacker should not be able to generate the signature of his favorite message given just the public key. This, however, is not a sufficiently strong notion of security. Typically, in fact, an attacker sees several signed message prior to attempting to forge a new one. In addition, some of the previously signed messages may have been chosen by him! (For instance, a notary public typically signs messages chosen by others.) Thus, not only should forge message be "hard," but also not "learnable". Let us now be more precise.

An *attacker* is a (possibly probabilistic) interactive algorithm A. An attack of A to a signature scheme (G, S, V) with security parameter ℓ consists of the following process. First, a pair of matching keys (pk, sk) is generated by running G on a random string of length ℓ. Then pk is provided as the only input to A. After receiving this input, A computes a query message m_1. This query is answered by providing A with the signature $\sigma_1 = S(sk, m_1)$. At this point A computes a second query message m_2, which is answered by providing A with the signature $\sigma_2 = S(sk, m_2)$. This interactive process continues until A halts outputting a pair of strings (m, σ). We say that A's attack is *successful* if (1) $m \notin \{m_1, m_2, \ldots\}$, and (2) $V(pk, m, \sigma) =$YES.

We say that the scheme (G, S, V) is *(GMR-)secure* if, for all constants c and for all sufficiently large security parameter ℓ, for every attacker A that (1) is described by a string of at most ℓ^c bits, and (2) halts within ℓ^c steps on every input of length ℓ, the probability that an attack of A against (G, S, V) with security parameter ℓ is successful is at most ℓ^{-c}.

Such signature schemes secure in the above sense were originally constructed by Goldwasser, Micali, and Rivest [GMR88] under standard complexity assumptions, such as the computational hardness of factoring integers. Subsequent works [BM92, NY89, Rom90] have progressively weakened the assumption, eventually showing that secure signature schemes can be constructed from any *one-way function*, in essence any function that is easy to compute but hard to invert.

3 Formal Model of the Problem

As discussed, our coding schemes envisage a stateful sender with a pair of matching public and secret keys. A stateful sender, on input a first message m_1 and its two keys, produces an encoding of the message together with a first state information, s_1. To encode the $i+1$st message, the sender utilizes, in addition to his two keys and the message itself, the previous state information s_i to produce the desired encoding and the new state information. Since in our construction the only requirement on such sequences of state information is that they strictly increase (according to some total order on strings), for simplicity we define a coding scheme for the special case in which the state information of the ith encoding is the integer i.

As usual, we consider coding schemes that work for arbitrarily long messages and have a single independent parameter k, denoting the message length.

Definition 3.1 (Public-Key Coding Scheme)**.** A public-key coding scheme $\mathcal{CS}$ consists of three deterministic, polynomial-time, Turing machines $\mathcal{G}$, $\mathcal{S}$, and $\mathcal{R}$, (respectively called the *generator*, the *sender*, and the *receiver*), a finite *alphabet* Σ, and two positive reals, a *security parameter* $\epsilon < 1$ and an *asymptotic rate* R: $\mathcal{CS} = (\mathcal{G}, \mathcal{S}, \mathcal{R}, \Sigma, \epsilon, R)$.

For all sufficiently large message lengths k, letting $\ell = \lfloor k^\epsilon \rfloor$ and $n = \lfloor k/R \rfloor$,

- On input an ℓ-bit string, $\mathcal{G}$ outputs a pair (pk, sk) of ℓ-bit strings, where pk is called the *public key* and sk the *secret key*.
- On input (1) an integer i less than 2^ℓ, the *counter*, (2) the secret key sk, and (3) a string $m_i \in \Sigma^k$, the *message*, algorithm $\mathcal{S}$ outputs $i+1$ (as the new counter) and a string $x_i = \mathcal{S}(i, sk, m_i) \in \Sigma^n$, referred to as an *encoding* of m_i.
- On input the public key pk and a string $r \in \Sigma^n$ (suppositively a corrupted version of an encoding of a message m_i), algorithm $\mathcal{R}$ outputs a string $m' \in \Sigma^k$ (hopefully the message m_i).

□

To define the error-correction rate of $\mathcal{CS}$, we need to define how our adversarial channel is entitled to "attack" a coding scheme. We start by defining purely syntactically what it means for an adversarial channel to be computationally bounded. To increase the flexibility of our adversary, we let it choose totally different strategies for different values of the message-length parameter k, possibly in a total non-constructive fashion. (By contrast, our senders and receivers are fully constructive: they apply the same efficient procedures to all possible messages/transmissions of all possible lengths.) Then, in order to maximize the relevance of our result, we endow a channel attacking a coding scheme with *very* broad powers (such as choosing the very messages to be transmitted in addition to choosing which errors to introduce). Of course, in any realistic setting, the channel may have only a subset of such powers. But since our solution is very simple and resilient to all such powers, we find no point in analyzing a less powerful adversary.

Definition 3.2 (Computationally Bounded Adversarial Channels, Attacks and Successes)**.** A *computationally bounded* adversarial channel $\mathcal{C}$ is a sequence of (stateful) possibly probabilistic algorithms, $\mathcal{C} = \mathcal{C}_1, \mathcal{C}_2, \ldots$, one for any value of the parameter k, such that: for some absolute constant $c > 0$, (1) $\mathcal{C}_k$ is described by at most k^c bits, and (2) on any input(s), $\mathcal{C}_k$ halts within k^c steps of computation.

An *attack* of a computationally bounded adversarial channel $\mathcal{C}_k$ on a public-key coding scheme $\mathcal{CS} = (\mathcal{G}, \mathcal{S}, \mathcal{R}, \Sigma, \epsilon, R)$ with message length k consists of the following process. First, G generates a pair of matching keys (pk, sk) on input a random string of length $\ell = k^\epsilon$. Then pk is provided as the only input to $\mathcal{C}_k$. At this point, the channel computes the first message $m_1 = \mathcal{C}_k(pk)$ to be transmitted, together with its first state information ψ_1. Then the sender produces an encoding $x_1 = \mathcal{S}(1, sk, m_1)$ of this message using his counter and secret key. At this point, the channel computes $(r_1, m_2, \psi_2) = \mathcal{C}_k(\psi_1, x_1)$, where r_1 is the first word received by the receiver, m_2 is the next message to be transmitted, and ψ_2 is its new state information. This process continues, that is at stage $i+1$ the channels computes $(r_i, m_{i+1}, \psi_{i+1}) = \mathcal{C}_k(\psi_i, x_i)$ and the sender computes $x_{i+1} = \mathcal{S}(i+1, sk, m_{i+1})$, until the channel halts upon reaching its prescribed bound k^c on the number of its computational steps.

For positive real $\rho < 1$, we say that the attack is *ρ-successful* if, for some stage i, (1) the Hamming distance between x_i and r_i is at most $\rho \cdot n$, and (2) the receiver makes a decoding error, that is, $\mathcal{R}(pk, r_i) \neq m_i$. □

Note that the channel's state information may, in particular, include the current value of the Sender's counter, as well as the sequence of all previously chosen messages, and their corrupted versions. Since the Receiver is deterministic, the channel can easily compute how each transmission is decoded.

Definition 3.3 (Computationally Secure Coding Scheme)**.** We say that the public-key coding scheme $\mathcal{CS}$ is *computationally secure against an error rate ρ* if, for all computationally bounded adversarial channels $\mathcal{C}$, for all positive constants c, all sufficiently large k, and every $e \leq \rho \cdot n$, the probability that an attack by $\mathcal{C}_k$ on $\mathcal{CS}$ with message length k is e-successful is at most k^{-c}. □

4 Our Computationally Secure Coding Scheme

4.1 Intuitive Description of Our Coding Scheme

Our scheme is perhaps best understood by a "try, error, and fix" approach, omitting precise details.

As a first try, for each message m, the sender transmits $x = C(m)$, a list-decodable encoding of m. An adversarial channel may however foil this simple scheme by finding a different message m', whose encoding is x', together with a string r sufficiently close to both x and x' so that, by list-decoding r, the receiver computes both m and m' as candidate messages. We do not know whether this can be done efficiently for every list-decodable code, but the current state of knowledge does not rule this out either. Accordingly, the safe thing to do is to assume that this first try does not work.

An attempt to fix the above problem consists of having the sender first computes σ, his own digital signature of m, and then transmits $x = C(m \circ \sigma)$, the list-decodable encoding of m concatenated with σ. This way, to be capable to carry out the above attack, an adversarial channel would need to find a string r that, list-decoded, yields both $m \circ \sigma$ and some $m' \circ \sigma'$ such that $m' \neq m$ and σ' is a valid sender's signature of m'. Since digital signature are hard to forge, a computationally bounded channel should not be able to find such a string r. However, let us argue that a stateful channel might be able to bypass this difficulty. In essence, the channel may not need to find, from scratch, a new message m' and be able to forge its digital signature σ'. Rather, the sender may have already sent $x' = C(m' \circ \sigma')$ in the past. In this case, the channels may simply record such transmission, and wait for an opportunity to use it: namely, when the sender transmits $x = C(m \circ \sigma)$ such that x and x' are sufficiently close. Again, such an occurrence cannot be ruled out and so the safe thing to do is to assume that this second try too does not work.

This leads us to a final fix, that we later on formalize and analyze. Namely, the sender keeps track of how many messages he has transmitted so far, using a counter i. Letting m be ith message he wants to send, the sender transmits the $x = C(m \circ i \circ \sigma)$, where σ is his digital signature of the *pair* (m, i). Let us see how this prevents the last problem, assuming for a moment that the receiver too keeps state information, and thus knows the current counter value. Assume that the sender has just transmitted $x = C(m \circ i \circ \sigma)$, that he had already transmitted $x' = C(m' \circ i' \circ \sigma')$ in the past, and that there is a string r sufficiently close to both x and x'. Thus, even if the channel causes the receiver to receive r rather than x, the receiver, upon to list-decoding r to compute $m \circ i \circ \sigma$ and $m' \circ i' \circ \sigma'$ as possible candidates, can use the current counter value to disambiguate between the two. Let us now show that he can correctly disambiguate without keep any state information, and thus without knowing what the current counter value should be. Notice that having sent the message m' prior to sending the message m, the sender actually transmitted $m \circ i \circ \sigma$ and $m' \circ i' \circ \sigma'$, where $i' < i$. Accordingly, if the receiver list-decodes r to get both $m \circ i \circ \sigma$ and $m' \circ i' \circ \sigma'$, it suffices for him to choose the unique decoding to be the one whose alleged counter is larger, in this example his choice would be $m \circ i \circ \sigma$. Notice that, to fool the receiver, the channel would have to find a different m', an integer $i' \geq i$, and a string r sufficiently close to both $x' = C(m' \circ i' \circ \sigma')$ and x. Furthermore σ' should be a valid sender's signature for the pair (m', i'). But to find all such strings, he must be able to *forge* the sender's signature on a never signed message (as opposed to utilizing a previously signed message). In fact, when his current counter has value i, the sender has never signed the pair (m', i'), either because $i' > i$, or because $i' = i$ but $m' \neq m$.

4.2 Formal Description of Our Coding Scheme

Given a signature scheme $\mathbb{S} = (G, S, V)$, a family of list-decodable codes $\mathbb{C} = \left\{C_k : \Sigma^k \to \Sigma^n\right\}$ with rate $R' > R$ and list-decoding algorithm $\mathcal{LD}$, our public-key coding scheme $\mathcal{CS} = (\mathcal{G}, \mathcal{S}, \mathcal{R}, \Sigma, \epsilon, R)$ works

as follows.

For each message length k, letting $\ell = \lfloor k^\epsilon \rfloor$ and $n = \lfloor (k + 2\ell)/R' \rfloor$:

- $\mathcal{G}$, on input a random ℓ-bit string s, computes the key pair $(pk, sk) = G(s)$ and sets the counter i to 0.
- $\mathcal{S}$, on input the counter $i < 2^\ell$ and a message $m \in \Sigma^k$, outputs $i + 1$ as the new counter, computes the digital signature $\sigma_i = S(sk, (m, i))$, and finally outputs $x_i = C_{k+2\ell}(m \circ i \circ \sigma_i)$ as the encoding of m. (Above $\circ$ is the concatenation operator, and i and σ_i are represented as elements of Σ^ℓ.)
- $\mathcal{R}$, on input the public key pk and a received word $r \in \Sigma^n$, works as follows (in essence "it list decodes r and outputs the most recently signed message"):

  1. Runs $\mathcal{LD}(r)$ to produce a list of strings Y.
  2. Computes the sublist $\mathcal{V}(Y)$ of *valid strings*, that is the sublist whose elements are the concatenation of three strings m, i and σ where
     — $m \in \Sigma^k$,
     — $i \in \Sigma^\ell$ represents an integer $< 2^\ell$, and
     — $\sigma \in \Sigma^\ell$ represents a valid signature of the pair (m, i), that is, $V(pk, (m, i), \sigma) =$YES.
  3. Computes the sublist $out\mathcal{V}(Y)$ of *most recent valid strings*, that is the elements $m \circ i \circ \sigma$ of $\mathcal{V}(Y)$ such that $i \geq i'$ for every other element $m' \circ i' \circ \sigma'$ in $\mathcal{V}(Y)$.
  4. Outputs the string m if $out\mathcal{V}(Y)$ contains a single element, of the form $m \circ i \circ \sigma$.
     Outputs *error* if $out\mathcal{V}(Y)$ is empty or has more than one element.

We refer to such $\mathcal{CS}$ as a *public-key coding scheme with underlying signature scheme* $\mathbb{S}$ *and list decodable code* $\mathbb{C}$.

4.3 Analysis of the Coding Scheme

We start by noticing that the rate of our code is indeed at least R, as claimed.

Proposition 4.1. *For every code $\mathbb{C}$ of rate $R' > R$, and signature scheme $\mathbb{S}$, and for every $\epsilon < 1$, the public-key coding scheme $\mathcal{CS}$, with underlying signature scheme $\mathbb{S}$ and list-decodable code $\mathbb{C}$, has rate R.*

Proof. We need to verify that for sufficiently large k, the length of the encoding of messages of length k is at most k/R. For every k, the length of the encoding is $n = (k + 2k^\epsilon)/R'$. To ensure this is at most k/R, we need $k(1/R - 1/R') \geq 2k^\epsilon/R'$ which holds if $k^{1-\epsilon} \geq \frac{2R}{R'-R}$, i.e., when $k \geq \left(\frac{2R}{R'-R}\right)^{1/(1-\epsilon)}$. *Q.E.D.*

We now analyze the robustness of our public-key coding scheme.

Lemma 4.2. *Let $\mathbb{S}$ be a GMR-secure digital signature scheme. Let $\mathbb{C}$ be a (ρ, L) efficiently list decodable code for some polynomially growing function $L(k)$. Then the public-key coding scheme $\mathcal{CS}$, with underlying signature scheme $\mathbb{S}$ and list-decodable code $\mathbb{C}$, is computationally secure against an error-rate ρ.*

Proof. We prove the lemma by contradiction. We assume that the coding scheme $\mathcal{CS}$ is not secure against error rate ρ, that is, that there exists a computationally bounded adversarial channel $\mathcal{C}$ whose attack on $\mathcal{CS}$ is ρ-successful. We obtain the desired contradiction by showing that this implies the existence of a successful attacker A on the signature scheme $\mathbb{S}$, so as to violate $\mathbb{S}$'s security assumption.

In essence, our proof consists of a *reduction* "from a successful adversarial channel $\mathcal{C}$ to a successful attacker A for the underlying signature scheme $\mathbb{S}$." The successful adversarial channel $\mathcal{C}$ is assumed by hypothesis, while the successful attacker A is constructed by us, using $\mathcal{C}$ as a subroutine.

Recall that, when attacking a signature scheme $\mathbb{S}$, an attacker is assumed to be given (1) a randomly generated public key pk, and (2) access to a signer who signs any message of the attacker's choice relative to pk. The essence of our reduction is that, on input pk, A attacks $\mathbb{S}$ by *simulating an attack of the channel $\mathcal{C}_k$ on $\mathcal{CS}$*. The difficulty of this approach is that, while by hypothesis the attacker has access to the signer, the channel presupposes a access to a sender, but there is no such a sender! Therefore, it will be A's responsibility to play the role of the sender (and the counter maintained by the sender) so that the channel's attack can be faithfully reproduced. Our analysis will then show that, if $\mathcal{C}_k$'s attack on $\mathcal{CS}$ is ρ-successful, then A is successful in forging a signature of a never-signed message.

Let us now give the details of how A operates.

Hypotheses

- The (ρ, L) efficient list decodable code is $\mathbb{C} = (E, \mathcal{LD})$;
- The signature scheme is $\mathbb{S} = (G, S, V)$;
- For a fixed parameter k, G has been run on input a random $\ell(k)$-bit string s and has generated the pair of matching keys (pk, sk);
- Attacker A is given input pk, oracle access to the signing function $S(sk, \cdot)$, and the subroutine $\mathcal{C}_k$;
- Without loss of generality, the channel decides whether to halt only at the end of a stage.

Initialization Stage

Attacker A runs the channel on input pk. The channel returns the first message-state pair $(m_1, \psi_1) = \mathcal{C}_k(pk)$. If the channel halts, so does the attacker A. Else, A sets the counter to 1 and proceeds to Stage 1.

Stage i

At the start of the stage, the overall state information consists of the current values of: (1) the counter, i, (2) the message to be sent, m_i, and (3) the channel state, ψ_i. The channel $\mathcal{C}_k$ now expects the "sender" to provide the encoding x_i of m_i. Accordingly, A asks for and receives the signature $\sigma_i = S(sk, (m_i, i))$, computes the encoding $x_i = E(k + 2\ell, m_i \circ i \circ \sigma_i)$, and return x_i to the channel. The channel responds by computing $(r_i, m_{i+1}, \psi_{i+1}) = \mathcal{C}_k(\psi_i, x_i)$, where r_i is the ith received string, m_{i+1} is the next message to be transmitted, and ψ_{i+1} is the next channel state. If the channel halts, so does the attacker A. Else, A A runs the receiver $\mathcal{R}$ on input pk and the received word r_i so as to compute:

(a) the list of strings Y,

(b) the sublist of valid strings $\mathcal{V}(Y)$,

(c) the sublist of most recent valid strings $out\mathcal{V}(Y)$, and

(d) the final output —either *error* or a message m— and decides whether to proceed to Stage $i+1$ based on the following cases (proven to be well defined in our analysis):

1. If $\delta(x_i, r_i) > \rho$, that is if the fraction of errors introduced by the channel exceeds the prescribed limit, then A sets the counter to $i + 1$ and proceeds to Stage $i + 1$.

2. (Else) If $\mathcal{R}$'s final output is m_i, then A sets the counter to $i+1$ and proceeds to Stage $i+1$.

3. (Else) If $\mathcal{R}$'s final output is a message m different from m_i, then A outputs the message-signature pair $((m,j),\sigma)$ such that $m \circ j \circ \sigma \in out\mathcal{V}(Y)$ and $j > i$, and HALTS.

4. (Else) If the final output is *error*, then A outputs $((m,j),\sigma)$ such that $m \circ j \circ \sigma \in out\mathcal{V}(Y)$ and either $j > i$ or $(j = i) \wedge (m \neq m_i)$, and HALTS.

Analysis

Note that Cases 1—4 are exhaustive. Let us now prove that, whenever the channel is ρ-successful, our attacker succeeds in forging a signature of a never-signed message. We start by noting that, if the channel is ρ-successful, then Cases 1 and 2 do not apply. In particular, this means that the fraction of errors introduced by the channel in the received string r_i, relative to the encoding x_i of the message m_i, is less than ρ. This implies that, when the receiver $\mathcal{R}$ lists decodes r_i, the list Y includes $m_i \circ i \circ \sigma_i$, where $\sigma_i = S(sk,(m_i,i))$ was requested and obtained by the attacker A. Furthermore, since σ_i is a valid signature of (m,i), the string $m_i \circ i \circ \sigma_i$ is also included in the sublist $\mathcal{V}(Y)$. Let us thus analyze Cases 3 and 4 *under this condition.*

Assume that Case 3 applies. By the definition of $\mathcal{R}$ this implies that the "most recent element" in the sublist $\mathcal{V}(Y)$ is unique and of the form $m \circ j \circ \sigma$, where σ is a valid signature, relative to the public key pk, of the pair (m,j). However, since $m_i \circ i \circ \sigma_i$ is also included in $\mathcal{V}(Y)$, it follows that $j > i$ and thus, by construction of A, that the pair (m,j) was never signed before. This implies that A's output, $((m,j),\sigma)$, is a forgery and that A's attack on the signature scheme has been successful.

Assume now that Case 4 applies. Since we are working under the condition that $\mathcal{V}(Y)$ contains the string $m_i \circ i \circ \sigma_i$, this implies that there multiple "most recent validly signed messages" in $out\mathcal{V}(Y)$, without loss of generality of the form $m^1 \circ j \circ \sigma^1$, $m^2 \circ j \circ \sigma^2$, etc., where $j \geq i$. We now distinguish two sub-cases: $j > i$ and $j = i$. In the first sub-case, A's output is of the form $((m^t,j),\sigma^t)$, a forgery of a never-signed message, because by construction A has only asked for the signatures of pairs whose second coordinate is at most i. In the second sub-case, there must exist a t such that $m^t \neq m_i$, and thus A's output is of the form $((m^t,i),\sigma^t)$, again a successful forgery, because by construction A only asks and receives the signature of a single pair whose second coordinate is i.

In sum: (1) the attacker A is well defined; (2) A is efficient because the additional computation it performs, relative to the computation of the adversarial channel, is trivial; and (3) A's attack on the signature scheme $\mathbb{S}$ is successful whenever the channel's attack on the public key coding scheme $\mathcal{CS}$ is ρ-successful.

This concludes the proof of our lemma.

Q.E.D.

Our main theorem, stated below, follows immediately from Proposition 4.1 and Lemma 4.2.

Theorem 4.3. *For every $\epsilon < 1$, $R' > R$, every GMR-secure signature scheme $\mathbb{S}$, and every (ρ, L)-efficiently list-decodable code $\mathcal{C}$ of rate R', the public-key coding scheme $\mathcal{CS} = (\mathcal{G},\mathcal{S},\mathcal{R},\Sigma,\epsilon,R)$, with underlying signature scheme $\mathbb{S}$ and list-decodable code $\mathbb{C}$, is computationally secure against error rate ρ.*

4.4 Concrete Instantiations

Here we instantiate our construction with concrete choices of list-decodable codes, yielding robust coding schemes for a variety of good parameter sets.

Binary coding schemes. To construct coding schemes over the binary alphabets $\Sigma_k = \{0, 1\}$ (for all k), we can use the poly-time constructible and efficiently list-decodable (concatenated) codes of Guruswami and Sudan [GS00]. By concatenating Reed-Solomon codes with inner codes of sufficiently large distance, they construct codes that are efficiently list decodable under $e = (\frac{1}{2} - \gamma) \cdot n$ errors (for any constant $\gamma > 0$) that have block length $\eta = O(\kappa/\gamma^8)$, i.e., the information rate $R = \Omega(\gamma^8)$. A more complicated construction that concatenates algebraic-geometry codes with the Hadamard code yields information rate $R = \Omega(\gamma^6 \log \frac{1}{\gamma})$. Further work by Guruswami *et al* [GHSZ02] gave codes of rate $\Omega(\gamma^4)$ having list size $O(\gamma^{-2})$. Most recently, Guruswami and Rudra [GR08] give codes of rate about $\Omega(\gamma^3)$ with list size $f(\gamma)$ for some bounded function f correcting $1/2 - \gamma$ fraction of errors. Any of these can be used to get our result for the setting of the binary channel.

Coding schemes for larger alphabets. With the use of larger alphabets (whose sizes grow with κ), we can list-decode under much larger numbers of errors.

The popular family of Reed-Solomon codes [RS60] was shown by Sudan [Sud97], and later Guruswami and Sudan [GS99], to be efficiently list decodable under high error rates. Specifically, there is a polynomial-time algorithm that can find all codewords within distance $\eta - \sqrt{\eta\kappa} = (1 - \sqrt{R})\eta$ of any word. The number of words in the list is at most η^2, and the alphabet size is $|\Sigma_\kappa| \approx \eta$.

The recent construction by Guruswami and Rudra [GR08] of *Folded Reed-Solomon* codes admits list-decoding algorithms with even better performance, correcting from $(1 - R - \gamma)\eta$ errors for any $\gamma > 0$. These codes are called "capacity-achieving," because the relationship between their list-decoding radius and information rate is (essentially) optimal.

5 Conclusions

Modeling channels as adversaries, as opposed to benign probabilistic processes, is obviously a safer choice. But such safety comes at a price: namely, the loss of information transmission rate. Realistically however, adversaries, physical processes and channels included, cannot be computationally unbounded. And once we realize this, we can actually leverage the power of cryptography to ensure that we can adopt the safe choice without degrading our information transmission rate.

References

[BM92] Mihir Bellare and Silvio Micali. How to sign given any trapdoor permutation. *J. ACM*, 39(1):214–233, 1992.

[DGL06] Yan Ding, Parikshit Gopalan, and Richard J. Lipton. Error correction against computationally bounded adversaries. Manuscript submitted to *Theory of Computing Systems*.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[Eli57] Peter Elias. List decoding for noisy channels. In *Wescon Convention Record*, Part 2, Institute of Radio Engineers (now IEEE), pages 94–104, 1957.

[GHSZ02] Venkatesan Guruswami, Johan Håstad, Madhu Sudan, and David Zuckerman. Combinatorial bounds for list decoding. *IEEE Transactions on Information Theory*, 48(5):1021–1034, 2002.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[GR08] Venkatesan Guruswami and Atri Rudra. Explicit codes achieving list decoding capacity: Error-correction with optimal redundancy. *IEEE Transactions on Information Theory*, 54(1):135–150, 2008.

[GS99] Venkatesan Guruswami and Madhu Sudan. Improved decoding of reed-solomon and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, 1999.

[GS00] Venkatesan Guruswami and Madhu Sudan. List decoding algorithms for certain concatenated codes. In *STOC*, pages 181–190, 2000.

[Gur03] Venkatesan Guruswami. List decoding with side information. In *IEEE Conference on Computational Complexity*, pages 300–, 2003.

[Ham50] Richard W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29:147–160, 1950.

[Lan04] Michael Langberg. Private codes or succinct random codes that are (almost) perfect. In *FOCS*, pages 325–334, 2004.

[Lip94] Richard J. Lipton. A new approach to information theory. In *STACS*, pages 699–708, 1994.

[MPSW05] Silvio Micali, Chris Peikert, Madhu Sudan, and David A. Wilson. Optimal error correction against computationally bounded noise. In *TCC*, pages 1–16, 2005.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *STOC*, pages 33–43, 1989.

[PV05] Farzad Parvaresh and Alexander Vardy. Correcting errors beyond the guruswami-sudan radius in polynomial time. In *Proc. FOCS*, pages 285–294. IEEE Computer Society, 2005.

[Plo60] Morris Plotkin. Binary codes with specified minimum distance. *IRE Transactions on Information Theory*, 6:445–450, September 1960.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *STOC*, pages 387–394, 1990.

[RS60] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *J. SIAM*, 8(2):300–304, June 1960.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, July 1948.

[Sud97] Madhu Sudan. Decoding of Reed Solomon codes beyond the error-correction bound. *J. Complexity*, 13(1):180–193, 1997.

[Woz58] J. M. Wozencraft. List decoding. *Quarterly Progress Report, Research Laboratory of Electronics, MIT*, 48:90–95, 1958.