Tolerant Combiners: Resilient Cryptographic Design

Amir Herzberg

Computer Science Department, Bar Ilan University, Ramat Gan, Israel
herzbea@cs.biu.ac.il
http://AmirHerzberg.com

Abstract. Cryptographic schemes are often designed as a combination of multiple component cryptographic modules. Such a combiner design is *tolerant* for a (security) specification if it meets the specification, provided that a sufficient subset of the components meet their specifications. The archtypical combiner is *cascade*, and we show that it is indeed a tolerant combiner for encryption schemes, under chosen plaintext attack, non-adaptive chosen ciphertext attack (CCA1) and (adaptive) replayable chosen ciphertext attack (RCCA), but *not* under the 'standard' adaptive chosen ciphertext attack (CCA2). We also analyze few other basic, folklore tolerant combiners, including the *parallel combiner* for one-way functions, and the *copy combiner* for integrity tasks such as Message Authentication Codes (MAC) and signature schemes. Cascade is also tolerant for the hiding property of commitment schemes, and the copy combiner is tolerant for the binding property, but neither provides tolerant for both properties.
We present (new) tolerant combiners for commitment schemes; these new combiners can be viewed as a composition of the cascade and the copy combiners. We prove tolerance of the composite combiners via a general Composition Lemma, possibly applicable for other tasks.
Our combiners are simple, efficient and practical. To ensure practicality, we use concrete security analysis and definitions, in addition to the simpler asymptotic analysis. Our definitions of security may be of independent interest.

Keywords: applied cryptography, tolerant cryptography, foundations of cryptography, concrete security

1 Introduction

Most cryptographic schemes do not have an unconditional proof of security. The classical method to establish security is by cryptanalysis, i.e. accumulated evidence of failure of experts to find weaknesses in the function. However, cryptanalysis is an expensive, time-consuming and fallible process. In particular, since a seemingly-minor change in a cryptographic function may allow an attack which was previously impossible, cryptanalysis allows only validation of specific functions and development of engineering principles and attack methodologies and

tools, but does not provide a solid theory for designing cryptographic functions. Indeed, it is impossible to precisely predict the rate of future cryptanalytical successes. Prudent designers are usually able to ensure security by using sufficient margins and conservative to allow for unexpected breakthroughs, e.g. [LV01]; however, there is often resistance to replace widely deployed standards which were not broken yet, 'just' since the safety margins are eroded.

Hence, it is desirable to design cryptographic schemes to be tolerant of cryptanalysis, failure of assumptions and other vulnerabilities, including trapdoors known to the designers (but not to users). A *tolerant combiner* is a combiner that combines several cryptographic modules, such that the combined mechanisms is secure even if some of the modules turn out to be insecure. In particular, the tolerant combiner remains secure following successful cryptanalysis of some of its modules, refutation of one or few of the assumptions underlying its security (e.g. the assumption that factoring is a hard problem), or attempts to exploit a vulnerability in some of the modules, due to implementation error, design errors, or an intentional trapdoor in the design or implementation. Tolerance does not imply unconditional-security; however, it would hopefully provide sufficient advanced-warning time to replace broken cryptographic modules.

Many cryptographic systems and combiners use redundant components in the hope of achieving tolerance. The most familiar such combiner is *cascade*. Cascading of cryptosystems is very natural; novices and experts alike believe that the cascade $\mathcal{E} \circ \mathcal{E}'$ of two encryption schemes $\mathcal{E}$, $\mathcal{E}'$ is at least as secure as the more secure of the two, hopefully even more secure than both. Indeed, cascading of cryptosystems has been a common practice in cryptography for hundreds of years.

However, so far, there are few publications analyzing tolerant cryptographic combiners. In [1], Asmuth and Blakely present a simple 'parallel' combiner of a randomized cryptosystem from two component ciphers, with the hope of achieving tolerance; proof of security was given only in [25]. A similar 'parallel' combiner for block ciphers appears in [31]. More attention was given to *cascading* of block ciphers . Even and Goldreich showed that keyed cascade ensures tolerance against message recovery attacks on block ciphers [19, Theorem 5], and conjectured that the result holds for other specifications of ciphers. Damgard and Knudsen [15] proved that it holds for security against key-recovery under chosen-plaintext attacks. Maurer and Massey [35] claimed that the proof in [19] holds only under the uninterestingly restrictive assumption that the enemy cannot exploit information about the plaintext statistics, but we disagree. We extend the proof of [19] and show that, as expected intuitively and in [19], keyed cascading provides tolerance to many confidentiality specifications, not only of block ciphers but also of other schemes such as public key and shared key cryptosystems.

Our proof of the tolerance of cascade encryption, holds for three definitions of security under indistinguishability test: the well-known notions of plaintext only attack and non-adaptive chosen ciphertext attack (CCA1), as well as the recently proposed relaxed definition of Replayable CCA (RCCA) [13]. We note that cascading does *not* provide tolerance for the well-known notion of adaptive

chosen ciphertext attack (CCA2). This demonstrates the importance of backing intuition with analysis and proof.

Tolerance is relevant to any cryptographic scheme, not just for confidentiality. In particular, it is widely accepted that the parallel combiner $g(x)\|f(x)$, using the same input x to both functions, ensures tolerance for several integrity properties, such as (several variants of) collision-resistant hashing as well as Message Authentication Codes (MAC) and digital signatures. We prove that the parallel combiner indeed provides tolerance for such integrity specifications. The parallel combiner is used, for tolerance, in practical designs and standards, e.g. in the W3C XML-DSIG specifications and in the TLS protocol [38].

We present few simple tolerant combiners for some basic cryptographic goals (specifications); further research is required to find tolerant combiners for other goals. In particular, Harnik et al. recenlty investigated robust combiners for oblivious transfer, and showed impossibility results for 'black box' combiners, as well as a robust combiner for oblivious transfer. We note that while they present their combiner 'from scratch', it also follows from our Composition Lemma, using composition structure E; see subsection 5.2. The Composition Lemma may be a useful methodology for other tasks.

Efficiency is critical for practical tolerant combiners; implementors will rarely be willing to tolerate non-negligible performance loss, 'just' in order to tolerate potential vulnerabilities in a cryptographic function. In fact, ignoring efficiency, we can ensure tolerance for many tasks, by using provable combiners of cryptographic mechanisms from few 'basic' cryptographic mechanisms such as one-way function, which have simple tolerant designs.

Specifically, the simple *parallel* combiner $f||f'(< x, x' >) = f(x)||f'(x')$ is tolerant for the one-way function specification. Namely, it is sufficient that one of $\{f, f'\}$ is a one-way function, to ensure that $f||f'$ is also a one-way function. Furthermore, there are reductions, probably secure under asymptotic (poly-time) definitions, of many cryptographic mechanisms from one-way functions and vice verse, e.g. pseudo-random generators [21, 29] and signature schemes [37]. It therefore follows that there are tolerant combiners for all of these tasks, albeit only for asymptotic (poly-time) definitions and involving substantial degradation in efficiency, including in the required security parameters (e.g., require absurd key and/or block sizes) [21, 30]. To quantify loss in security and efficiency due to the combiners, we use concrete security measures [7, 8].

Our contributions. We identify and define cryptographic tolerance as a criteria for cryptographic specifications, and define tolerant combiners. Additional contributions include:

- *Precise analysis of the security of several 'folklore' combiners.* In particular, we show that cascade encryption indeed ensures tolerance as long as each component encryption has fixed output length (for fixed length input), and for several variants of indistinguishability including replayable chosen ciphertext attack (RCCA), but not for the 'regular' adaptive chosen ciphertext attack (CCA2) specification. We note that the 'multiple encryption' combiner of [16] seems to ensure tolerant encryption for CCA2, but at significant overhead (ciphertext length more than doubles), which may be unacceptable for

many applications. Replayable CCA is weaker than 'full' adaptive chosen ciphertext attack, but it is arguably a sufficient requirement for most applications, in particular it allows the practical 'feedback only CCA' attacks of [6, 32]. One interpretation of our results may be as an additional argument in favor of using the RCCA criteria, rather than the (stronger or too strong) CCA2 criteria.

- *Efficient, practical combiners for commitment schemes.* To our knowledge, these are the first provably-secure tolerant combiners of general cryptographic functions, beyond the folklore combiners, and few additional cryptographic combiners proven secure based on validity of either of two (specific) 'hardness' assumptions, e.g. [42].
- *Composition Lemma and Methodology.* Cryptographic combiners are usually studied in isolation; however, sometimes one combiner is good for ensuring some properties, e.g. confidentiality, while another is good for other properties, e.g. integrity. We define compositions of combiners, to combine the benefits of different combiners (when possible), and present a generic composition based on a simple combinatorial object (composition structure). Finally, we use the generic composition and two simple composition objects to compose the cascade and parallel combiners, creating two new efficient composite combiners for commitment schemes. The composition lemma may be useful for designing and analyzing tolerant combiners for other tasks; in particular, the design of [27] can be viewed as application of our 'composition E'.
- *Generalized definitions for encryption.* We generalize existing poly-time and concrete security definitions of encryption; the resulting definition is quite general, and may therefore be of independent interest.

2 Definitions: Cryptographic Schemes and Tolerant Combiners

For simplicity and generality, we allow cryptographic schemes to be any function, regardless of a specific computational model. To model cryptographic schemes which consist of multiple functions, we simply view the choice of the particular function as an additional input, specified using dot notation. For example, if $\mathcal{E}$ is an encryption scheme, we let $\mathcal{E}.E$ and $\mathcal{E}.D$ denote the encryption and decryption functions of $\mathcal{E}$, respectively. Similarly, to model randomized and/or stateful mechanisms, we simply use additional inputs for random bits and/or state, respectively. In particular, this allows us to model interactive cryptographic mechanisms, by specifying the complete transcript of the interaction as an input.

It is convenient to assume, without loss of generalization, that every cryptographic scheme has a special input which we call the *security parameter* $k \in \mathbb{N}$. Intuitively, the security parameter selects the level of security desired; for example, a larger security parameter implies longer keys and more computational resources. Since the security parameter is input to every scheme (or every function in the scheme), we usually do not explicitly write it as an input. Some

definitions, e.g. One Way Functions [21], do not use a security parameter, but use the length of the input, as an 'implicit' security parameter.

The security parameter is often used to bound the time complexity and other resources. In particular, we say that scheme $\mathcal{S}$ is *polynomial time*, or simply *poly-time*, if its computational time is bounded by a polynomial in the security parameter k. Similarly, we say that scheme $\mathcal{S}$ has *time complexity bound* $\mathcal{S}.t : \mathbb{N} \rightarrow \mathbb{N}$ if its computation time is bounded by $\mathcal{S}.t(k)$, where k is the security parameter.

We define the *time complexity predicate* $Time(T)$ for scheme $\mathcal{S}$ as follows: let $Time(T)[\mathcal{S}] = True$ when $\mathcal{S}.t(k) \leq T(k)$. The function $T : \mathbb{N} \rightarrow \mathbb{N}$ is a bound on the time complexity, as a function of the security parameter k. Namely, the time complexity of scheme $\mathcal{S}$ is bounded by $T(k)$. Given a set of schemes $\mathcal{S}_1, \ldots \mathcal{S}_p$, let $Time(T)[\mathcal{S}_1, \ldots \mathcal{S}_p] = 1$ when for all $i \in \{1, \ldots p\}$ holds $Time(T)[\mathcal{S}_i]$.

We are interested in specifications (properties) of cryptographic schemes, including concrete security specifications and asymptotic security specifications. We define specifications simply as binary predicates over the set of (cryptographic) schemes. We say that scheme $\mathcal{S}$ *satisfies* specifications s when $s(\mathcal{S}) = 1$.

We say that a mapping c of p schemes $\mathcal{S}_1, \ldots \mathcal{S}_p$ into a single scheme (or function) $c(\mathcal{S}_1, \ldots \mathcal{S}_p)$ is a *combiner of plurality* p. Combiner c is *tolerant* if $c(\mathcal{S}_1, \ldots \mathcal{S}_p)$ satisfies some specification s' as long as a sufficient subset of $\mathcal{S}_1, \ldots \mathcal{S}_p$ satisfy specifications $s_1, \ldots s_p$, respectively. Often, all specifications are identical, i.e. $s = s_1 = \cdots = s_p$ and also often $s = s'$.

To complete this definition, we need to identify the sufficient subset of $\mathcal{S}_1, \ldots \mathcal{S}_p$. Following works on secret-sharing, we define two variants of tolerance: one based on threshold t $(0 \leq t < p)$, and the other based on general access structure $\Lambda \subseteq P(\{1, \ldots p\})$ (where Λ is a set of subsets of $\{1, \ldots p\}$). In addition, we often require that *all* of the candidate schemes $\mathcal{S}_i$ satisfy some minimal specifications b, such as bounds on their complexities.

Definition 1. *Let c be a combiner of plurality p, and let $s', b, s_1, \ldots, s_p$ be predicates over (cryptographic) schemes. Combiner c is t-tolerant for $(s_1, \ldots s_p) \rightarrow s'$, with threshold $t < p$, if $s'(c(\mathcal{S}_1, \ldots \mathcal{S}_p))$ holds provided $\sum_{i=1}^{p} s_i(\mathcal{S}_i) \geq p - t$. combiner c is Λ-tolerant for $(s_1, \ldots s_p) \rightarrow s'$, with access structure $\Lambda \subseteq P(1, \ldots p)$, if $s'(c(\mathcal{S}_1, \ldots \mathcal{S}_p))$ holds provided for some $\lambda \in \Lambda$ holds $(i \in \lambda) \rightarrow s_i(\mathcal{S}_i) = 1$. combiner c is t-tolerant (Λ-tolerant) with prerequisite b if it is t-tolerant (respectively Λ-tolerant) provided that $b(s_1, \ldots s_p) = 1$. If c is 0-tolerant for $(s_1, \ldots s_p) \rightarrow s'$ then we say that c preserves $(s_1, \ldots s_p) \rightarrow s'$. If c is t-tolerant for $(s_1, \ldots s_p) \rightarrow s'$ and $s = s_1 = \ldots = s_p$, then we say that c is (t, p)-tolerant combiner for $s \rightarrow s'$; if $t = 0$ then we say that c* preserves *$s \rightarrow s'$. If $s = s'$ then we say that c is a (t, p)-tolerant combiner for (or preserves) s.*

In this work, we present and prove the tolerance of combiners for several cryptographic schemes. We mostly use the 'standard' definitions of these schemes, e.g. One-Way Functions and commitment schemes from [21], and encryption, signature, and message authentication codes (MAC) schemes from [22].

The definitions in [21, 22] are all using the asymptotic, polynomial-time security specifications. In the following two subsections, we extend the definitions

for encryption and commitment schemes in several ways, most notably allowing also the (arguably more practical) 'concrete security' specifications. We believe that our definitions improve somewhat compared to earlier concrete-security definitions, e.g. for encryption schemes in [7].

2.1 Encryption Schemes.

An encryption scheme $\mathcal{E}$ consists of three functions $<KG, E, D>$ (for key generation, encryption and decryption, respectively). We define encryption schemes with respect to four ensembles of domains: **EK, DK, M** and **C**, for the encryption keys, decryption keys, plaintext messages and ciphertexts, respectively. The ensembles are indexed by the *security parameter* $k \in \mathbb{N}$. We denote the ensembles using bold font e.g. **EK**, and the k^{th} member of the ensemble by the corresponding letter and subscript e.g. EK_k. Formally, the domains and their ensembles are properties of the encryption scheme, and should be identified with respect to it, e.g. $\mathcal{E}.\mathbf{M}$ and $\mathcal{E}.M_k$; however, we abuse notation and omit the identification of the scheme, i.e. write simply **M** and M_k, when we discuss a single encryption scheme (and therefore no confusion is possible).

The key generation function $\mathcal{E}.KG$ accepts as input a random string $r_{KG} \in \{0,1\}^k$, and its output is a pair of keys: $< e, d >= \mathcal{E}.KG(r_{KG}) \in EK_k \times DK_k$ for encryption and decryption, respectively. For symmetric encryption, we simply use $e = d$ and **E**=**D**. We again use dot notation to refer to particular key, i.e. $< \mathcal{E}.KG.e(r_{KG}), \mathcal{E}.KG.e(r_{KG}) >= \mathcal{E}.KG(r_{KG})$.

We use subscripts to denote keys, and the random input to the encryption function. Encryption of message m, where $m \in M_k$, using encryption key $e \in EK_k$ and randomness $r \in \{0,1\}^*$, is $\mathcal{E}.E_{e,r}(m) \in C_k$. The decryption function $\mathcal{E}.D$ accepts as input ciphertext $c \in C_k$, and private decryption key $d \in DK_k$, and returns a message $m' \in M_k$ or a failure indicator $\perp$.

We require all encryption schemes to satisfy the *correctness requirement* (for any $k \in \mathbb{N}$): $\mathcal{E}.D_d(\mathcal{E}.E_{e,r}(m)) = m$, for any $m \in M_k, r \in \{0,1\}^*$ and $r_{KG} \in \{0,1\}^k$ with $e = E.KG.e(r_{KG}), d = E.KG.d(r_{KG})$.

Security of Encryption: Indistinguishability Experiment. To define security for encryption schemes, we use (and extend) the quantitative 'indistinguishability experiment' approach of [7]. We first define the experiment. Our definition extends the definition of [7] as follows:

- We define the experiment for *both* symmetric (shared-key) and asymmetric (public-key) cryptosystems. The only required difference is a flag, denoted ϕ; only when $\phi =$ ASYM, the adversary receives the (public) encryption key $e = \mathcal{E}.KG.e(r_{KG})$.
- Most definitions, e.g. of [7], define the output of the experiment as a boolean value (WIN or FAIL). However, we allow the use of multiple success criteria, allowing both asymptotic, poly-time analysis and concrete security analysis. Therefore, we define the output of the experiment as the entire execution trace, $EX(\mathcal{E}, A, \phi, k)$, which is a function of the given encryption scheme

$\mathcal{E}$, adversary A, symmetric/asymmetric cryptosystem flag ϕ and security parameter k. We later define success criteria.

– At the post-selection attack phase, we allow the adversary to request decryption of every ciphertext except the challenge c^*, as in the 'standard' adaptive chosen ciphertext attack. However, we also allow the adversary to ask for the weaker, 'relaxed decryption' service, which returns the special value REPLAY if the decrypted plaintext is one of the two plaintexts selected by the adversary. This allows us to define criteria for the experiment, capturing the adaptive chosen ciphertext attack (CCA2) definition of e.g. [7]), as well as the weaker 'replayable CCA' attack of [13].

Definition 2 (Indistinguishability Experiment). *Let $\mathcal{E}$ be an encryption scheme, A (Adversary) be an interactive Turing machine, $k \in \mathbb{N}$, and flag $\phi \in \{\text{SYM, ASYM}\}$. Then $EX(\mathcal{E}, A, \phi, k)$ is a random variable, produced by tracing all events in the following execution:*

Key Generation: $r_{KG} \in_R \{0,1\}^k$; $e = \mathcal{E}.KG.e(r_{KG})$; $d = \mathcal{E}.KG.d(r_{KG})$;
Expose public key: `If` $\phi =$ ASYM `then send` $e, 1^k$ *to A,* `else send` *(only)* 1^k *to A*
Select: `Repeat:`

1. `If` *A outputs* $<$ ENCRYPT$, m >$ *where* $m \in M_k$, `then return` $c = \mathcal{E}.E_{e,r}(m)$ *to A, where* $r \in \{0,1\}^*$;
2. `If` *A outputs* $<$ DECRYPT$, c >$ *where* $c \in C_k$, `then return` $m = \mathcal{E}.D_d(c)$ *to A;*

`Until` *A outputs* $<$ SELECT$, m_0, m_1 >$ *where* $m_0,\ m_1 \in M_k$.
Encrypt: `Return` $c^* = \mathcal{E}.E_{e,r}(m_b)$ *to A, where* $r \in \{0,1\}^*$ *and* $b \in_R \{0,1\}$;
Guess: `Repeat:`

1. `If` *A outputs* $<$ ENCRYPT$, m >$ *where* $m \in M_k$, `then return` $c = \mathcal{E}.E_{e,r}(m)$ *to A, where* $r \in \{0,1\}^*$;
2. `If` *A outputs* $<$ DECRYPT$, c >$ *and* $c \in C_k/c*$, `then return` $m = \mathcal{E}.D_d(c)$ *to A;*
3. `If` *A outputs* $<$ R-DECRYPT$, c >$ *where* $c \in C_k$, `then let` $m = \mathcal{E}.D_d(c)$. `If` $m \in \{m_0, m_1\}$ `then return` REPLAY *to A;* `else return** m *to A.*

`Until` *A outputs* $<$ GUESS$, b' >$ *where* $b' \in \{0,1\}$.

The experiment as defined above is quite general. It allows for both asymmetric (public key) and symmetric (shared key) cryptosystems. It also allows multiple variant of attacks, including: chosen-plaintext attack (CPA); 'lunchtime'/non-adaptive CCA (aka CCA1); adaptive chosen ciphertext attacks (aka CCA2); and the weaker replayable (or relaxed) CCA (RCCA). Finally, the experiment allows different measures of security, including the 'classical' asymptotic, poly-time security, as well as the more practical concrete security. We next define appropriate definitions of security for encryption schemes.

The security definitions refer to different properties of the execution. We next define these properties, referring to an execution $X = EX(\mathcal{E}, A, \phi, k)$. We refer to values of the variables used in execution X, e.g. the random choice b and the adversary's guess b', using dot notation, i.e. $X.b$ and $X.b'$, respectively. We define other properties, also using dot notation, as follows:

- $X.win = True$ if the guess is correct, i.e. $X.b' = X.b$, and $False$ otherwise.
- $X.t$ is the total time spent by adversary during X
- For $stage \in \{\text{SELECT,GUESS}\}$ and $test \in \{E, D, rD\}$, let $X.q(stage, test)$ be the number of outputs (tests) of type $test$ of A in stage $stage$.

Asymptotic, Poly-Time Security of Encryption We now present the 'classical', asymptotic polynomial time secure encryption definitions, using the general Indistinguishability experiment of Definition 2. We allow the adversary to use polynomial time and to use a polynomial number of each type of test queries, possibly restricting queries for the 'weaker' notions (cf. to CCA2), i.e. CPA, CCA1 and RCCA.

Definition 3. *Let $\phi \in \{\text{SYM, ASYM}\}$ and $\psi \in \{\text{CPA, CCA1, CCA2, RCCA}\}$. Encryption scheme $\mathcal{E}$ is an* IND-ψ polytime secure ϕ-cryptosystem*, if for every polytime adversary A and strictly positive polynomial $\epsilon : \mathbb{N} \to \mathbb{R}^+$, for sufficiently large $k \in \mathbb{N}$ holds*

$$Pr[X.win \wedge \Psi(X, \psi) | X = Ex(\mathcal{E}, A, \phi, k)] < \frac{1}{2} + \epsilon(k)$$

Where $\Psi(X, \psi)$ is defined as follows:

- $\Psi(X, \text{CPA})$ *holds if* $q(stage, t) = 0$ *for* $stage \in \{\text{SELECT,GUESS}\}$, $t \in \{D, rD\}$
- $\Psi(X, \text{CCA1})$ *holds if* $q(\text{GUESS}, t) = 0$ *for* $t \in \{D, rD\}$
- $\Psi(X, \text{RCCA})$ *holds if* $q(\text{GUESS,D}, t) = 0$
- $\Psi(X, \text{CCA})$ *is always* $True$.

Concrete Security of Encryption Schemes We next define concrete security for encryption schemes (cryptosystems).

Definition 4. *Let $\phi \in \{\text{SYM, ASYM}\}$, $Q = \{\{\text{SELECT,GUESS}\} \times \{E, D, rD\} \to \mathbb{N}\}$ and $\epsilon : \mathbb{N} \times \mathbb{N} \times Q \to (0, \frac{1}{2})$.*

*Encryption scheme $\mathcal{E}$ is an $\epsilon-$*IND secure ϕ-cryptosystem*, if for every adversary A and every $k \in \mathbb{N}$, $t \in \mathbb{N}$ and $q \in Q$ holds*

$$Pr[X.win \wedge X.q \leq q \wedge X.t \leq t | X = Ex(\mathcal{E}, A, \phi, k)] < \frac{1}{2} + \epsilon(k, t, q)$$

2.2 Commitment Schemes.

A (non-interactive) commitment scheme $\mathcal{C}$ consists of four functions[1] $\mathcal{C}.KG$, $\mathcal{C}.C$, $\mathcal{C}.D$, $\mathcal{C}.V >$ (for Key Generation, Commit, Decommit and Validate, respectively).

[1] Most existing definitions of commitment schemes, e.g. in [21] use a function to recover the message, instead of our 'validate' function. However, as a result, the combiners use long decommitment strings, which contain the original message. This may hide inefficiency in the design of a combiner; by explicitly using the message as separate input, we can compare the actual overhead of combiners.

We define commitment schemes with respect to four ensembles: **PK, M, CT** and **DT**, for the Public Keys, Messages, Commitment Tags and Decommitment Tags, respectively. The ensembles are indexed by the *security parameter* $k \in \mathbb{N}$, e.g. PK_k is the domain of public keys of the commitment scheme, for security parameter k.

The key generation function $\mathcal{C}.KG$ accepts as input a random string in $\{0,1\}^k$, and outputs a public commitment key $pk \in PK_k$. The commit and decommit functions $\mathcal{C}.C, \mathcal{C}.D$ have three inputs each: a message $m \in M_k$, a public commitment key $pk \in PK_k$ and randomness r, and their respective outputs are: a commitment tag $c = \mathcal{C}.C_{pk,r}(m) \in CT_k$ and a decommitment tag $d = \mathcal{C}.D_{pk,r}(m) \in DT_k$.

The validate function $\mathcal{C}.V$ has the same inputs as the commit and decommit functions, plus the commitment and decommitment tags (c, d respectively), and outputs $True$ if only if c, d are a correct commitment and decommitment values for m. All commitment schemes must satisfy the following *correctness requirement*: $\mathcal{C}.V_{pk}(m, \mathcal{C}.C_{pk,r}(m), \mathcal{C}.D_{pk,r}(m)) = True$, for any $k \in \mathbb{N}$, $m \in M_k$, $r, r_{KG} \in \{0,1\}^k$ such that $pk = \mathcal{C}.KG(r_{KG}) \in PK_k$.

Commitment schemes have two main properties (specifications): a confidentiality (indistinguishability) property, called *hiding*, and an integrity property, called *binding*.

Hiding (Indistinguishability) Specification for Commitment Schemes. We first define the hiding (indistinguishability) specifications for commitment schemes, which is rather similar to the indistinguishability specifications presented earlier for encryption schemes. Intuitively, the (polytime) hiding specification is that no probabilistic polynomial time (PPT) adversary can distinguish between the commitments of any two messages of its choice, even if allowed to choose the public commitment key.

In order to turn this intuition into a definition, we first define an indistinguishability experiment for commitment schemes.

Definition 5 (Indistinguishability Experiment for Commitment schemes). *Let $\mathcal{C}$ be a commitment scheme, A (Adversary) be an interactive Turing machine and $k \in \mathbb{N}$. Then $IndEx(\mathcal{C}, A, k)$ is a random variable, produced by tracing all events in the following execution:*

Adversary Selects Public Key and Messages: *Adversary A outputs public key $pk \in PK_k$ and two messages, $m_0,\ m_1 \in M_k$.*
Experiment Selects Bit and Commit: *Select $b \in_r \{0,1\}$ and $r \in \{0,1\}^k$.* `Return` $c^* = \mathcal{C}.C_{pk,r}(m_b)$ *to A.*
Adversary Guesses: *A outputs $b' \in \{0,1\}$.*

The security definitions refer to different properties of the execution. We next define these properties, referring to an execution $X = IndEx(\mathcal{E}, A, k)$. As for encryption, we refer to values of the variables used in execution $X = IndEx(\mathcal{E}, A, k)$ using dot notation. We also define $X.win$ as a flag to indicate

an execution where $b = b'$, and $X.t$ as the total time spent by adversary during X.

We now present asymptotic, polynomial time 'hiding' specification for commitment schemes, using the Indistinguishability experiment of Definition 5.

Definition 6. *Commitment scheme $\mathcal{C}$ is polytime hiding, if for every polytime adversary A and strictly positive polynomial $\epsilon : \mathbb{N} \to \mathbb{R}^+$, for sufficiently large $k \in \mathbb{N}$ holds*

$$Pr[X.win | X = IndEx(\mathcal{C}, A, k)] < \frac{1}{2} + \epsilon(k)$$

We next give concrete security specifications for the hiding (indistinguishability) property of commitment schemes.

Definition 7. *Let $\epsilon : \mathbb{N} \times \mathbb{N} \to (0, \frac{1}{2})$. Commitment scheme $\mathcal{C}$ is $\epsilon-$*Hiding*, if for every adversary A and every $k \in \mathbb{N},\ t \in \mathbb{N}$ holds*

$$Pr[X.win \wedge X.t \leq t | X = IndEx(\mathcal{C}, A, k)] < \frac{1}{2} + \epsilon(k, t)$$

Binding Specification for Commitment Schemes. We now define the binding, integrity specification for commitment schemes. Here is where we use the public commitment key; this key is picked by the 'recipient' of the commitment, and therefore the adversary, acting now as a potentially malicious 'sender' of the commitment, has to use a random public commitment key, rather than choosing the worst possible key.

Intuitively, given (random) public commitment key pk, every (PPT) adversary A has negligible probability of finding a collision, i.e. values c, d, d', m, m' s.t. $\mathcal{C}.V_{pk}(m, c, d) = \mathcal{C}.V_{pk}(m', c, d') = True$ (notice the commitment c is the same!). In order to turn this intuition into a definition, we first define a *collision experiment for commitment schemes.*

Definition 8 (Collision (Binding) Experiment for Commitment schemes). *Let $\mathcal{C}$ be a commitment scheme, A (Adversary) be an interactive Turing machine and $k \in \mathbb{N}$. Then $ColEx(\mathcal{C}, A, k)$ is a random variable, produced by tracing all events in the following execution:*

Select Public Key: *Select $r_{KG} \in \{0,1\}^k$ and give to the adversary $pk = \mathcal{C}.KG(r_{KG})$.*

Adversary outputs collision: *Adversary A output $m_0 \neq m_1 \in M_k$ and $r_0, r_1 \in \{0,1\}^k$ such that $\mathcal{C}.C_{pk,r_0}(m_0) = \mathcal{C}.C_{pk,r_1}(m_1)$.*

We now present asymptotic, polynomial time 'binding' specification for commitment schemes. We define $X.collision$ as a flag that is *true* if and only if X completes with adversary outputting a valid collision, as validated in the last step of the experiment. We also use $X.t$ as the total time spent by adversary during X.

Definition 9. *Commitment scheme $\mathcal{C}$ is polytime binding, if for every polytime adversary A and strictly positive polynomial $\epsilon : \mathbb{N} \to \mathbb{R}^+$, for sufficiently large $k \in \mathbb{N}$ holds*

$$Pr[X.collision|X = BindEx(\mathcal{C}, A, k)] < \frac{1}{2} + \epsilon(k)$$

We next give concrete security specifications for the binding (integrity) property of commitment schemes.

Definition 10. *Let $\epsilon : \mathbb{N} \times \mathbb{N} \to (0, \frac{1}{2})$. Commitment scheme $\mathcal{C}$ is $\epsilon-$*Binding*, if for every adversary A and every $k \in \mathbb{N},\ t \in \mathbb{N}$ holds*

$$Pr[X.collision \wedge X.t \leq t|X = BindEx(\mathcal{C}, A, k)] < \frac{1}{2} + \epsilon(k, t)$$

We conjecture that the combiner we present (later) is also tolerant for several other variants of commitment schemes, such as the relaxed binding of [3], trapdoor commitments [43], and chameleon hash functions [33].

3 The Cascade Combiner and its Tolerance

The most basic tolerant combiner is probably *cascade*. We begin by discussing cascading of functions with a single input and output, such as one-way functions and (key-less) hash functions, namely $f \circ g(x) = f(g(x))$. In the following subsections we discuss cascading of keyed schemes.

3.1 Simple Cascade of Keyless Cryptographic Functions

Consider any two functions $g : D_g \to R_g$, $f : D_f \to R_f$ s.t. $R_g \subseteq D_f$. The simple cascade of f and g, denoted $f \circ g$, is a combiner of plurality 2 defined as $c \circ (f, g) = f \circ g(x) = f(g(x))$.

Unfortunately, simple cascade rarely ensures tolerance, and often does not even preserve cryptographic specifications. So far, we found simple cascade ensures tolerance only to the one-way function specification as defined in [21]. Even that is true only with a prerequisite requirement $perm(f)$, which is true only if f is a permutation when restricted to input domains $\{0,1\}^l$ for some length l.

Lemma 1. *Simple cascade of two functions is...*

1. *1-tolerant with prerequisite perm for specifications OWF.*
2. *Not (even) 0-tolerant for specifications OWF and $WCRHF$, as defined in [21].*

Proof: Claim 1 follows from a trivial reduction argument.

To prove claim 2, let h be a OWF and/or $WCRHF$. Let $g(x) = h(x)||0^{|h(x)|}$ and $f(x) = \begin{cases} 0 & \text{if } x = y0^{|x|/2} \\ h(x) & \text{otherwise} \end{cases}$. Trivially, both f and g are OWF and/or $WCRHF$, respectively, yet $f \circ g$ is neither OWF not $WCRHF$; in fact, $f \circ g(x) = 0$ for every x. □

3.2 Cascade Encryption Is Tolerant

The cascade encryption, i.e. cascade of two[2] encryption schemes $\mathcal{E}', \mathcal{E}''$, is denoted $\mathcal{E}' \circ \mathcal{E}''$ and defined as follows. *Notation:* For convenience we explicitly write the inputs and outputs to the cascade (or any combiner) as a tuple of inputs or outputs when appropriate, e.g. $< r', r'' >$ to denote the pair of two random inputs (r' to $\mathcal{E}'$ and r'' to $\mathcal{E}''$).

- $\mathcal{E}' \circ \mathcal{E}''.KG.e(< r', r'' >) =< \mathcal{E}'.KG.e(r'), \mathcal{E}''.KG.e(r'') >$
- $\mathcal{E}' \circ \mathcal{E}''.KG.d(< r', r'' >) =< \mathcal{E}'.KG.d(r'), \mathcal{E}''.KG.d(r'') >$
- $\mathcal{E}' \circ \mathcal{E}''.E_{<e',e''>,<r',r''>}(m) = \mathcal{E}'.E_{e',r'}(\mathcal{E}''.E_{e'',r''}(m))$
- $\mathcal{E}' \circ \mathcal{E}''.D_{<d',d''>}(c) = \mathcal{E}''.D_{d''}(\mathcal{E}'.D_{d'}(c))$

For simplicity, we require that the domains of $\mathcal{E}'$ and of $\mathcal{E}''$ are *compatible*, i.e. that $\mathcal{E}''.C_k \subseteq \mathcal{E}'.M_k$. We define the predicate *compatible* to be true when applied to a pair of compatible encryption schemes. This is sufficient to cap the time complexity of the cascade, as the sum of the time complexity of the two component cryptosystems. This trivial observation is stated in the following lemma.

Lemma 2. *Let $\mathcal{E}', \mathcal{E}''$ be a pair of encryption schemes such that $\mathcal{E}''.C_k \subseteq \mathcal{E}'.M_k$. Then $\mathcal{E}' \circ \mathcal{E}''.t(k) \leq \mathcal{E}'.t(k) + \mathcal{E}''.t(k)$.* □

We now investigate the security and tolerance of cascade encryption. As noted in the introduction, cascade encryption is an ancient, widely-deployed technique, usually in the hope of improving security - e.g., providing tolerance to weaknesses of one of the two cascaded encryption schemes. Is this secure? This depends on the adversary capabilities ('attack model'). Cascade encryption is not tolerant for adaptive chosen ciphertext attack (CCA2); simply consider $\mathcal{E}''$ which ignores the least significant bit of the ciphertext, allowing adversary to decrypt the challenge ciphertext (by flipping the LSb and invoking the decryption oracle). However, as [3, 13] argued, this 'attack' is so contrived, that it may indicate that CCA2 is overly restrictive, rather than a problem with cascade encryption. In [3], the authors present a slightly weaker definition, gCCA, but we do not think cascade is tolerant under that definition, either. However, the following lemma shows that cascade encryption is tolerant under the more relaxed *Replayable CCA (RCCA)* definition of [13].

Lemma 3. *Cascade encryption is a $(1, 2)$-tolerant combiner for specifications* IND-ψ polytime secure ϕ-cryptosystem*, for $\psi \in \{CPA, CCA1, RCCA\}$.*

Proof: follows immediately from Lemma 4 below. □

We next show that cascade encryption is tolerant under our 'concrete security' specification, as in Definition 4. Our proof uses essentially the same (simple) reduction argument as in [19].

[2] Extension to arbitrary number of schemes is trivial.

Lemma 4. *Cascade encryption is a* $(1,2)$*-tolerant combiner for specifications* $\epsilon-$IND secure $\phi-$cryptosystem $\rightarrow \epsilon^*-$IND secure $\phi-$cryptosystem *with prerequisites* $Time(T)$ *and compatible, for every* $\epsilon(k,t,q)$ *as in Definition 4 and* $\epsilon(k,t,q) = \epsilon^*(k, t + max(q)T(k), q)$.

Proof: Let $\mathcal{E}', \mathcal{E}''$ be a pair of encryption schemes such that $Time(T)$ and *compatible* hold, i.e. $\mathcal{E}''.C_k \subseteq \mathcal{E}'.M_k$. Suppose to the contrary that there exists adversary A, security parameter k and time complexity t such that

$$Pr[X.win \wedge X.t \leq t | X = EX(\mathcal{E}' \circ \mathcal{E}'', A, \phi, k)] \geq \frac{1}{2} + \epsilon^*(k,t,q)$$

We use A as a subroutine of adversaries A' and A'', showing that $\mathcal{E}'$ and $\mathcal{E}''$, respectively, are not $\epsilon-$IND secure $\phi-$cryptosystem. The design of A' and A'' is similar; we therefore only present A'.

Adversary A' is trying to 'win' the indistinguishability experiment for $\mathcal{E}'$, by running the indistinguishability experiment for A, but using E' as a 'black box'. Specifically, A' operates as follows, following the steps in Definition 2:

Key Generation: A' generates keys for E'', i.e. $r_{KG} \in_R \{0,1\}^k$; $e'' = \mathcal{E}.KG.e(r_{KG})$; $d'' = \mathcal{E}.KG.d(r_{KG})$;

Expose public key: `If` $\phi =$ ASYM `then send` $< e', e'' >, 1^k$ to A, `else send` (only) 1^k to A

Select and Guess phases: During these phases, A' needs to respond to the encryption and decryption requests of A. However, this is easy to do using the corresponding services of E', and of course E'' (for which A' has selected the keys). For example, when A outputs $<$ ENCRYPT$, m >$ where $m \in M_k$, then A' asks for encryption c' of m by E', i.e. $c' = E'_{e',r}(m)$, and then computes and returns its encryption by E''.

Encrypt: When A outputs, after select phase, the choice $<$ SELECT$, m_0, m_1 >$ where $m_0,\ m_1 \in M_k$, then A' simply outputs the same choice. When receiving the response $(c^* = \mathcal{E}.E_{e,r}(m_b))$, then A' encrypts this by E'' and returns the result to A.

Output: When A outputs $<$ GUESS$, b' >$, then A' outputs the same guess.

Obviously, A wins if and only if A' wins. But A cannot win with probability better than $\frac{1}{2} + \epsilon^(k,t,q)$. Since A' only performs one more computation of E'' for each query, the claim follows. □

Cascading is a natural candidate combiner for many cryptographic mechanisms; we now define and investigate tolerance of cascade of commitment and MAC/Signature schemes.

3.3 Cascade Commitment

We define cascade commitment $\mathcal{C}' \circ \mathcal{C}''$, i.e. cascade of two commitment schemes $\mathcal{C}', \mathcal{C}''$, as follows. We again wrote inputs and outputs as tuples.

- $\mathcal{C}' \circ \mathcal{C}''.KG(< r', r'' >) =< \mathcal{C}'.KG(r'), \mathcal{C}''.KG(r'') >$

- $\mathcal{C}' \circ \mathcal{C}''.C_{<pk',pk''>,<r',r''>}(m) = \mathcal{C}'.C_{pk',r'}(\mathcal{C}''.C_{pk'',r''}(m))$
- $\mathcal{C}' \circ \mathcal{C}''.D_{<pk',pk''>,<r',r''>}(m)$
  $=< \mathcal{C}'.D_{pk',r'}(\mathcal{C}''.C_{pk'',r''}(m)), \mathcal{C}''.D_{pk'',r''}(m), \mathcal{C}''.C_{pk'',r''}(m) >$
- $\mathcal{C}' \circ \mathcal{C}''.V_{<pk',pk''>}(m, c', < d', d'', c' >) = \mathcal{C}''.V_{pk''}(m, c'', d'') \wedge \mathcal{C}'.V_{pk'}(c'', c', d')$.

Similarly to cascade encryption, we require that the domains of $\mathcal{C}'$ and of $\mathcal{C}''$ are *compatible*, i.e. that $\mathcal{C}''.CT_k \subseteq \mathcal{C}'.M_k$. The predicate *compatible* is true when applied to a pair of compatible commitment schemes. This is sufficient to cap the time complexity of the cascade, as the sum of the time complexity of the two component commitments, however note that in a single application of the cascade decommit function, we need to apply both the $\mathcal{C}''$ commit and decommit functions. This trivial observation is stated in the following lemma.

Lemma 5. *Let $\mathcal{C}', \mathcal{C}''$ be a pair of compatible commitment schemes. Then $\mathcal{C}' \circ \mathcal{C}''.t(k) \leq \mathcal{C}'.t(k) + 2 \cdot \mathcal{C}''.t(k)$.* □

As the following lemma shows, cascade ensures the privacy (hiding) property of commitment schemes, but only preserves the integrity (binding) property.

Lemma 6. *Cascade commitment is a $(1,2)$-tolerant combiner for specification polytime hiding. Cascade commitment is a $(1,2)$-tolerant combiner for specifications $\epsilon-$Hiding $\rightarrow \epsilon^*-$Hiding with prerequisites $Time(T)$ and compatible, for every $\epsilon(k,t)$ as in Definition 7 and $\epsilon(k,t) = \epsilon^*(k, t + T(k))$.*

Proof: A simplified version of the proof of Lemma 4. □

Finally, the following lemma shows that cascade only preserves the integrity (binding) specification of commitment schemes.

Lemma 7. *Cascade commitment preserves specifications polytime hiding and $\epsilon-$Binding $\rightarrow \epsilon^*-$Bindgin with prerequisites $Time(T)$ and compatible, for every $\epsilon(k,t)$ as in Definition 7 and $\epsilon(k,t) = \epsilon^*(k, t + T(k))$.*

Proof (sketch): Every collision of $\mathcal{C}' \circ \mathcal{C}''$, gives a collision of (at least) one of $\mathcal{C}'$, $\mathcal{C}''$. □

Similarly, we next show that cascade also preserves, but does not ensure tolerance, for other integrity properties, specifically of MAC and Signature schemes.

3.4 Cascading Preserves Unforgeability of MAC and Signature Schemes

We define cascade $\mathcal{S} \circ \mathcal{S}'$ of two MAC or Signature schemes $\mathcal{S}, \mathcal{S}'$, as follows. We again write inputs and outputs as tuples.

1. $\mathcal{S} \circ \mathcal{S}'.KG.s(< r, r' >) =< \mathcal{S}.KG.s(r), \mathcal{S}'.KG.s(r') >$
2. $\mathcal{S} \circ \mathcal{S}'.KG.v(< r, r' >) =< \mathcal{S}.KG.v(r), \mathcal{S}'.KG.v(r') >$
3. $\mathcal{S} \circ \mathcal{S}'.S_{<s,s'>,<r,r'>}(m) =< \mathcal{S}.S_{s,r}(m)(\mathcal{S}'.S_{s',r'}(m)) >$
4. $\mathcal{S} \circ \mathcal{S}'.V_{<v,v'>}(\sigma, < m, \sigma' >) = \mathcal{S}.V_v(\sigma, \sigma') \wedge \mathcal{S}'.V_{v'}(\sigma', m)$

The following lemma shows that cascade (only) preserves the existential unforgeability specification of MAC and signature schemes, as defined in [22]. These are poly-time (asymptotic) security specifications, but a similar concrete-security formalization easily follows and is omitted.

Lemma 8. *Cascade of MAC and signature schemes is* 0*-tolerant for (i.e. preserves) the* existential unforgeability under adaptive chosen message attack *specification. Cascade of MAC and signature schemes preserves the existential unforgeability specification.*

Proof (sketch): Every forgery of $\mathcal{C}' \circ \mathcal{C}''$, gives a forgery of (at least) one of $\mathcal{C}'$, $\mathcal{C}''$. □

4 Parallel and Copy Combiners and their Tolerance

We now consider another important family of combiners, which are parallel applications of two or more cryptographic functions or schemes. Parallel combiners may use the same input to all functions, use different parts of the input to each function, or use some combination of the inputs to create the input to each function, often involving XOR or secret-sharing. Similarly, the output of some parallel combiners is simply the concatenation of the outputs of each function, while others 'merge' the outputs, by XOR or secret-sharing.

4.1 Parallel Combiner for OWF

Possibly the simplest parallel combiner 'splits' the input among several functions, and concatenates the result. In particular, the *Parallel* combiner for two keyless functions f, f' is defined as $f||_p f'(< x, x' >) =< f(x), f(x') >$. This trivial combiner is tolerant for One-Way Functions specifications, using two or more functions.

Lemma 9. *The Parallel combiner* $f||_p f'(< x, x' >) =< f(x), f(x') >$ *is tolerant for the OWF specifications.*

Proof Sketch: Suppose that for some $< x, x' >$, there is a non-negligible probability that the adversary is able to find preimage $< y, y' >$ given only $f||_p f'(< x, x' >)$. Since $< y, y' >$ are preimages, it follows that $f(y) = f(x)$, $f'(y') = f'(x')$. Hence, this gives a preimage also to x (and x'), i.e. there is a non-negligible probability that the adversary is also able to invert f and f'. □

4.2 Copy Combiner for Integrity Specifications of Hash Functions, Commitment, MAC and Signature Schemes

The *Copy* combiner is also trivial and well-known, but it is very practical and widely deployed. Here, the input to the combiner is 'copied' and used as input to each of the components; and the output is simply the concatenation of the

output of all components. This simple, folklore combiner provides tolerance for the integrity properties of collision-resistant hash functions, signature and MAC schemes and commitment schemes.

Let us first define the copy combiner for keyless functions, e.g. (weakly collision resistant) hash functions. The copy combiner of single-input (keyless) functions f, g is denoted as $f||g$, and defined as $f||g(x) =< f(x), g(x) >$. When the functions have inputs for random bits and/or keys, these are selected independently for the two functions, and the parallel combiner is $f_{k,r}||g_{k',r'}(x) =< f_{k,r}(x), g_{k',r'}(x) >$.

The copy combiner of two Signature or MAC schemes $\mathcal{S}, \mathcal{S}'$, denoted $\mathcal{S}||\mathcal{S}'$, is defined as follows. The definitions and proofs extend trivially to arbitrary number of schemes.

- $\mathcal{S}||\mathcal{S}'.KG(< r, r' >) =< \mathcal{S}.KG(r), \mathcal{S}'.KG(r') >$
- $\mathcal{S}||\mathcal{S}'.S_{<s,s'>,<r,r'>}(m) =< \mathcal{S}.S_{s,r}(m), \mathcal{S}'.S_{s',r'}(m) >$
- $\mathcal{S}||\mathcal{S}'.V_{v,v'}(m, < \sigma, \sigma' >) = \mathcal{S}.V_v(m, \sigma) \wedge \mathcal{S}'.V_{v'}(m, \sigma')$

Similarly, the copy combiner of two commitment schemes $\mathcal{C}, \mathcal{C}'$, denoted $c_{||}(\mathcal{C}, \mathcal{C}') = \mathcal{C}||\mathcal{C}'$, is defined as follows.

- $\mathcal{C}||\mathcal{C}'.KG(< r, r' >) =< \mathcal{C}.KG(r), \mathcal{C}'.KG(r') >$
- $\mathcal{C}||\mathcal{C}'.C_{<pk,pk'>,<r,r'>}(m) =< \mathcal{C}.C_{pk,r}(m), \mathcal{C}'.C_{pk',r'}(m) >$
- $\mathcal{C}||\mathcal{C}'.D_{<pk,pk'>,<r,r'>}(m) =< \mathcal{C}.D_{pk,r}(m), \mathcal{C}'.D_{pk',r'}(m) >$
- $\mathcal{C}||\mathcal{C}'.V_{<pk,pk'>}(m, < c, c' >, < d, d' >) = \mathcal{C}.V_{pk}(m, c, d) \wedge \mathcal{C}.V_{pk}(m, c, d)$

As the following lemma shows, the copy combiner ensures tolerance for many integrity properties / specifications, but clearly is quite bad for privacy.

Lemma 10. *The copy combiner as defined above is...*

1. *Tolerant for the* existential unforgeability under adaptive chosen message attack *specification of Signature and MAC schemes.*
2. *Tolerant for the* polytime binding *specifications of commitment schemes.*
3. *Preserving, but* not *tolerant, for the 'confidentiality' specifications* polytime hiding *of commitment schemes, and the* IND-CCA1 , IND-CPA, IND-CCA2 *and* IND-RCCA *polytime secure specifications, of (symmetric and asymmetric) encryption schemes.*
4. Not *tolerant, for (the 'confidentiality' specifications)* OWF.

4.3 The XOR Combiner for Encryption

Another classical tolerant combiner, originally proposed in [1] for encryption schemes, takes two inputs: a message (plaintext) and a random bit string of the same length, and applies one function to the random string, and the other function to the exclusive-OR of the message with the random string. Namely, the simple *XOR* combiner for two keyless functions f, f' is defined as $f \oplus f'(< m, x >) =< f(m \oplus x), f'(x) >$; generalization to more than two functions is trivial.

The definition for XOR combiner for encryption schemes $\mathcal{E}, \mathcal{E}'$, is similar:

- $\mathcal{E} \oplus \mathcal{E}'.KG.e(< r, r' >) =< \mathcal{E}.KG.e(r), \mathcal{E}'.KG.e(r') >$
- $\mathcal{E} \oplus \mathcal{E}'.KG.d(< r, r' >) =< \mathcal{E}.KG.d(r), \mathcal{E}'.KG.d(r') >$
- $\mathcal{E} \oplus \mathcal{E}'.E_{<e,e'>,<r,r',x>}(m) =< \mathcal{E}.E_{e,r}(x), \mathcal{E}'.E_{e',r'}(x \oplus m) >$
- $\mathcal{E} \oplus \mathcal{E}'.D_{<d,d'>}(< c, c' >) = \mathcal{E}'.D_{d'}(c') \oplus \mathcal{E}.D_d(c)$

The XOR combiner provides tolerance for the (relatively weak) notions of CPA and CCA1, but (trivially) not for the stronger notions of CPA2 or even RCCA.

Lemma 11. *The XOR combiner of encryption schemes is* $(1, 2)$*-tolerant for specifications* IND-ψ polytime secure ϕ-cryptosystem*, for* $\psi \in \{CPA, CCA1\}$ *and* $\phi \in \{\text{SYM, ASYM}\}$. .

Proof (sketch): Given an adversary $A^{\oplus}$ that can distinguish $\mathcal{E} \oplus \mathcal{E}'$, we construct an adversaries A, A' to distinguish $\mathcal{E}$ and $\mathcal{E}'$, respectively. The design of A, A' is simple, since we deal with only CPA and CCA1 attacks; namely, encryption and decryption done only *before* the messages $m_0,\ m_1$ are selected. Therefore, at the select phase, A and A' can easily answer all the queries of $A^{\oplus}$, by selecting arbitrary key pair for $\mathcal{E}'$ or $\mathcal{E}$, respectively.

When $A^{\oplus}$ selects the messages $m_0,\ m_1$, the adversaries A and A' simply provide the same messages in their experiments. When A (A') receives ciphertext c^* to distinguish in its experiment, it gives to $A^{\oplus}$ the ciphertext $< c^*, \mathcal{E}'.E_{e',r}(0) >$ (respectively, $< \mathcal{E}.E_{e,r}(0), c^* >$). By outputing the guess of $A^{\oplus}$, the adversaries A and A' have the same chance of winning as $A^{\oplus}$. □

4.4 Sharing Combiner for Tolerant Commitment

In the sharing combiner, the inputs to each component commitment scheme are *shares* of the input to the combiner. A *secret sharing scheme* is a pair of algorithms $< Share, Reconstruct >$. The $Share$ algorithm accepts a message m as input, and outputs n secret values $s_1, \ldots, s_n$ which we call shares; it is randomized, i.e. it also accepts some random input r. For convenience, let $Share_{i,r}(m, n)$ denote the i^{th} output of $Share$ on input m, number of shares n and randomness r. *Reconstruct* is a deterministic algorithm which takes n shares, $s'_1, \ldots, s'_n$, some of which may have the special value $\perp$ (for a missing share), and outputs a message m' (or $\perp$ for failure). The correctness property is that for every message m and randomness r holds $m = Reconstruct(Share_{1,r}(m, n), \ldots, Share_{n,r}(m, n))$.

Secret sharing schemes support different *thresholds*, for tolerating exposure or corruption of shares. In particular, in our case, we are interested in the following two thresholds. First, secret sharing schemes have a privacy threshold, t_p, which determines the maximum number of shares which reveal 'no information' about the message m. Second, they have a soundness threshold t_s, which determines the minimum number of correct shares which ensures it is impossible to recover an *incorrect* message $m' \neq m$ (and $m' \neq \perp$).

For simplicity, we present the sharing combiner (sc) for ensuring tolerance from three candidate commitment schemes, $\mathcal{C}_1, \mathcal{C}_2$ and $\mathcal{C}_3$, and using an arbitrary secret sharing scheme $< Share, Reconstruct >$ with $n = 3, t_p = 1, t_s = 2$,

e.g. Shamir's scheme [40]. Generalizations allowing threshold to $t_p > 1$ insecure components (by using $2t_p + 1$ components and shares) are straightforward. We use the notation $s_i = Share_{i,r}(m, n)$.

$sc(\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3).KG(< r_1, r_2, r_3 >) =$
$\quad =< \mathcal{C}_1.KG(r_1), \mathcal{C}_2.KG(r_2), \mathcal{C}_3.KG(r_3) >$
$sc(\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3).C_{<ck_1,ck_2,ck_3>,<r,r_1,r_2,r_3>}(m) =$
$\quad =< \mathcal{C}_1.C_{ck_1,r_1}(s_1), \mathcal{C}_2.C_{ck_2,r_2}(s_2), \mathcal{C}_3.C_{ck_3,r_3}(s_3) >$
$sc(\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3).D_{<ck_1,ck_2,ck_3>,<r,r_1,r_2,r_3>}(m) =$
$\quad =< \mathcal{C}_1.D_{ck_1,r_1}(s_1), \mathcal{C}_2.D_{ck_2,r_2}(s_2), \mathcal{C}_3.D_{ck_3,r_3}(s_3), s_1, s_2, s_3 >$
$sc(\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3).V_{<ck_1,ck_2,ck_3>}(m, < c_1, c_2, c_3 >, < d_1, d_2, d_3, s_1, s_2, s_3 >) =$
$= \{True \text{ iff } (m = Reconstruct(s_1, s_2, s_3)) \wedge ((\forall_{i=1,2,3})\mathcal{C}_i.V_{ck_i}(s_i, c_i, d_i))\}$

The tolerance of the sharing combiner follows easily from the properties of secret sharing schemes. Essentially, the sharing combiner is a hybrid or generalization of the copy combiner and of the XOR combiner.

Lemma 12. *The Share-Parallel-Concat (sc) combiner of* $2t + 1$ *commitment schemes is* t*-tolerant for both the* Binding *and the* Hiding *specifications, for every* $t \geq 1$. □

Comment. In most practical commitment schemes, decommitment requires mainly the original message, and the additional decommitment strings d_i are quite short. However, the sharing combiner uses long decommitment string; specifically, the decommitment includes $< d_1, d_2, d_3, s_1, s_2, s_3 >$. Typcically, e.g. using Shamir's secret sharing scheme [40], each share is as long as the message; namely the decommitment string of the sharing combiner is three times as long as the original message. This may be substantial overhead for many applications. The scheme we present in the next section avoids this overhead.

Comment. By using robust secret sharing and other tools, [16] achieve tolerant combiner for the IND-CCA2 specification of encryption schemes. However, their combiner is very wasteful in the length of the ciphertext, which may rule it unacceptable in most applications; we expect cascade would remain the preferred tolerant combiner for encryption in practice (although it is not tolerant for CCA2).

5 Composition of Combiners, and Efficient Tolerant Commitment

Often, we may want to combine multiple combiners, e.g. to ensure tolerance to multiple specifications. We restrict our attention to simple compositions of two combiners. In the first subsection we present two ways to compose the cascade combiner (tolerant for *hiding*) and the copy combiner (tolerant for *binding*), resulting in efficient tolerant combiners for commitment schemes (ensuring both hiding and binding specifications). In the second subsection, we generalize these results, by defining a composition as a mapping of (two) combiners, and presenting a general Composition Lemma, deriving the compositions for commitment schemes, which we presented in the first subsection, as a special case.

5.1 'Composite' Combiners for Tolerant yet Efficient Commitments

The sharing combiner provides tolerant design for commitment schemes, but results in a long decommitment string (three times the original message), which may be problematic for many applications. Can we construct efficient tolerant commitment schemes, with short decommitment (and commitment) strings? In this subsection we show two such combiners, with different tradeoffs. Both of these combiners are compositions of the cascade and copy combiners. This builds on the following key properties:

- Cascade is tolerant for hiding and preserves binding
- Copy is tolerant for the binding and preserves hiding.

It therefore makes sense to create a combiner which applies both cascade and hiding. In particular, we can use *four* candidate commitment schemes, $\mathcal{C}_{11}, \mathcal{C}_{12}, \mathcal{C}_{21}$, and $\mathcal{C}_{22}$, cascading $\mathcal{C}_{11}$ and $\mathcal{C}_{12}$ and connecting this in parallel to the cascade of $\mathcal{C}_{21}$ and $\mathcal{C}_{22}$. We call the result the D combiner, after its 'shape', and define it as follows. We use the notation $\mathcal{C}_{ij}(m) = \mathcal{C}_{ij}.C_{k_{ij},r_{ij}}(m)$, $\mathcal{D}_{ij}(m) = \mathcal{C}_{ij}.D_{k_{ij},r_{ij}}(m)$, $\mathcal{V}_{ij}(m) = \mathcal{C}_{ij}.V_{k_{ij}}(m, c_{ij}, d_{ij})$, $R =< r_{11}, r_{12}, r_{21}, r_{22} >$, $K =< k_{11}, k_{12}, k_{21}, k_{22} >= D.KG(R)$.

- $D.KG(R) =< \mathcal{C}_{11}.KG(r_{11}), C_{12}.KG(r_{12}), C_{21}.KG(r_{21}), C_{22}.KG(r_{22}) >$
- $D.C_{K,R}(m) =< \mathcal{C}_{12}(\mathcal{C}_{11}(m)), \mathcal{C}_{22}(\mathcal{C}_{21}((m)) >$
- $D.D_{K,R}(m)$
  $=< \mathcal{D}_{11}(m), \mathcal{C}_{11}(m), \mathcal{D}_{12}(\mathcal{C}_{11}(m)), \mathcal{D}_{21}(m), \mathcal{C}_{21}(m), \mathcal{D}_{22}(\mathcal{C}_{21}(m)) >$
- $D.V_K(m, < c_{12}, c_{22} >, < d_{11}, c_{11}, d_{12}, d_{21}, c_{21}, d_{22} >)$
  $= \mathcal{V}_{11}(m) \wedge \mathcal{V}_{21}(m) \wedge \mathcal{V}_{12}(c_{11}) \wedge \mathcal{V}_{22}(c_{21})$

The D combiner is quite efficient in computation times (each operation requires one operation from each of the four candidate commitment schemes), and in the size of the commit and decommit strings (commit size is twice that of the candidate commitment schemes, and decommit size consist of four decommitments plus two commitments). In particular, in the size of the commit and decommit strings, it substantially improves upon the sharing combiner; this may be important for many applications.

However, the D combiner has one significant drawback: it uses four component commitment schemes for 1-tolerance, while the sharing combiner requires only three candidate schemes for 1-tolerance. We can fix this by using only three commitment schemes, but using each of them *twice*, by connecting in parallel three cascades of two schemes each; we call this the E combiner.

The E combiner uses only three candidate commitment schemes, $\mathcal{C}_0, \mathcal{C}_1$ and $\mathcal{C}_2$. Specifically, the E combiner is the copy combiner applied to three cascades of pairs of candidate schemes: $\mathcal{C}_0 \circ \mathcal{C}_1$, $\mathcal{C}_1 \circ \mathcal{C}_2$, and $\mathcal{C}_2 \circ \mathcal{C}_0$.

We next state the tolerance of the D and E combiners. The proof is given in the next subsection.

Lemma 13. *The D (E) combiner of $2t+2$ (respectively, $2t+1$) commitment schemes is t-tolerant for* Binding *and* Hiding *specifications.* □

5.2 The Composition Lemma for Combiners

We now generalize the idea of combining multiple combiners, as in the previous subsection, to arbitrary combiners and specifications. We still limit our attention to compositions of two combiners. Such compositions accept as input two combiners c and c' and produce a composite combiner denoted $c' \circ_I c$, where I is a mapping of the 'candidate functions' to the combiners. We present few simple, and useful, compositions. First, we need to define the relevant mappings I and the composition for given I.

Let c be a combiner of plurality p over F which is t-tolerant for $s \rightarrow s'$, and let c' be a combiner of plurality p' over F which is t'-tolerant for $s' \rightarrow s$". Let p° denote the plurality of the composition of c and c'; namely the input to the composite combiner is an ordered set f of p° functions, $f[i] \in F$. The composite combiner $c' \circ_I c$ first applies c to p' sets of p functions each, and then applies c' to the p' resulting functions. The composition is defined by the selection of the p functions input to each of the p' applications of the c combiner, namely by a mapping $I : \{1, \ldots, p\} \times \{1, \ldots, p'\} \rightarrow \{1, \ldots, p^\circ\}$, where $I_i[j]$ identifies the j^{th} function input to the i^{th} c combiner. Given I, the I-composition of c' and c, denoted $c' \circ_I c$, is

$$c' \circ_I c\left(f[1], ..., f[p^\circ]\right) = c'\left(c\left(f[I_1(1)], ..., f[I_1(p)]\right), ..., c\left(f[I_{p'}(1)], ..., f[I_{p'}(p)]\right)\right)$$

Consider cascade compositions of threshold-tolerant combiners. The following lemma shows that the security of the I-composition of two threshold-tolerant combiners, depends on a simple combinatorial property of mappings I. Consider mapping $I : \{1, \ldots, p\} \times \{1, \ldots, p'\} \rightarrow \{1, \ldots, p^\circ\}$ and some set $T \subseteq \{1, \ldots, p^\circ\}$ (of 'weak input functions'). Let $G_i(I, T) = \{I_i[j] | j = 1, \ldots, p\} - T$, i.e. values $I_i[j]$, for some j, which are *not* in T; think of $G_i(I, T)$ as the 'good selections' of I_i. Let $G(I, T)[t] = \{i \ s.t. \ |G_i(I, T)| \geq p - t\}$. We say that I is a (good) (t, t', t°)-*threshold-composition-structure* if for every $T \subseteq 1, \ldots, p^\circ$ s.t. $|T| \leq t^\circ$ holds: $|G(I, T)[t]| \geq p' - t'$.

Lemma 14. *Let $I : \{1, \ldots, p\} \times \{1, \ldots, p'\} \rightarrow \{1, \ldots, p^\circ\}$ be a (good) (t, t', t°)-threshold-composition-structure. Let c be a combiner of plurality p over F which is t-tolerant for $s \rightarrow s'$. Let c' be a combiner of plurality p' over F which is t'-tolerant for $s' \rightarrow s$". Then $c' \circ_I c$, is a combiner of plurality p° over F which is t°-tolerant for $s \rightarrow s$".*

Proof: Consider any set f of p° functions, $f[i] \in F$, and assume that $p^\circ - t$ of them satisfy specification s. Namely, for some set $\{i_j\}$ of $p^\circ - t$ indexes holds $s(f[i_j]) = 1$. We need to prove that for every choice $T \subseteq \{1, \ldots, p^\circ\}$ of up to t° functions in f which do not satisfy s, the function resulting from applying composed combiner $c \circ_I c'$ to $\{f[1], \ldots, f[p^\circ]\}$ satisfies s". Namely, we need to prove that s"$(c' \circ_I c(f[1], \ldots, f[p^\circ])) = 1$. Let $f'[1], \ldots, f'[p']$ denote the p' intermediate functions, i.e. $f'[i] = c(f[I_i(1)], \ldots, f[I_i(p)])$; hence $c' \circ_I c(f[1], \ldots, f[p^\circ]) = c'(f'[1], \ldots, f'[p'])$.

If $i \in G(I, T)[t]$, namely $|G_i(I, T)| \geq p - t$, then for at least $p - t$ of the functions $f[I_i(1)], \ldots, f[I_i(p)]$ holds $s(f[I_i(j)]) = True$. Since c is t-tolerant for $s \rightarrow$

s', it follows that $s'(f'[i]) = True$, for every $i \in G(I,T)$. Since c' is t'-tolerant for $s' \rightarrow s$", it follows that: $s"(c' \circ_I c(f[1], \ldots, f[p^\circ])) = s"(c'(f'[1], \ldots, f'[p'])) = 1$. □

We now present two simple threshold cascade compositions derived from the above lemma, by presenting two simple composition structures:

- Composition structure $D : 0,1 \times 0,1 \rightarrow 0,1,2,3$ defined as $D_i[j] = 2i + j$ for $i, j \in 0, 1$.
- Composition structure $E : 0,1 \times 0,1,2 \rightarrow 0,1,2$ defined as $E_i[j] = i + j \pmod 3$ for $i \in 0,1,\ j \in 0,1,2$.

By simply checking the combinatorial definition of (t, t', t°)-threshold-composition-structure we get:

Lemma 15. *D and E are both (good) $(0,1,1)$ and $(1,0,1)$ threshold-composition-structures.* □

From the two Lemmas, we get:

Lemma 16. *Let c, c'_D, c'_E be combiners of plurality 2, 2 and 3 respectively. If c is t-tolerant for $s \rightarrow s'$ where $t \in 0,1$, and $c'_D,\ c'_E$ are $(1-t)$-tolerant for $s' \rightarrow s"$, then $c'_D \circ_D c$ and $c'_E \circ_E c$ are both 1-tolerant for $s \rightarrow s"$.* □

We can now prove Lemma 13. We present the proof for $t = 1$; the composition structures and proofs for $t > 1$ are similar.

Proof of Lemma 13 Let c be c_c, i.e. c is the cascade combiner; and let $c'_D,\ c'_E$ be $c_{||}$, i.e. the copy-parallel-concat combiner, both for commitment schemes. Notice that $D = c_{D'} \circ_D c,\ E = c_{E'} \circ_E c$. The claim follows immediately from Lemmas 6, 10 and 16. □

We note that the same Composition Lemma and structures can also be used, in a very similar manner, to derive the $(1,3)$-Tolerant combiner for Oblivious Transfer from [27]; furthermore, from the composition, we can clearly also extend this result to derive $(t, 2t+1)$-Tolerant combiner for Oblivious Transfer, for any $t \in \mathbb{N}$.

6 Conclusions and Open Questions

We presented simple, efficient and practical tolerant combiners for some of the most important and practical cryptographic mechanisms, including encryption, signature/MAC and commitment schemes. For encryption, MAC and signature schemes, we simply proved the security of the (very efficient) 'folklore' combiners; for commitment schemes, we present new combiners which are simple compositions of the folklore cascade and parallel (specifically, copy-parallel-concat) combiners. We also present definitions for tolerant combiners and compositions, and some basic yet useful results regarding compositions of combiners.

We believe that efficient tolerant combiners are an important requirement from practical cryptographic primitives; put differently, we should prefer specifications with an efficient tolerant combiner. We presented efficient tolerant

combiners for several of the important primitives (and specifications) of modern cryptography. However, for others, we did not find (yet?) a (reasonably efficient) tolerant combiner. This calls for additional research, to distinguish between specifications with efficient tolerant design, vs. specifications that do not have an efficient tolerant design (and possibly, to find alternate specifications which are sufficient for most applications/scenarios). In particular, following earlier versions of this work, tolerant combiners were found for Oblivious Transfer and Key Agreement in [27], and for IND-CCA2 encryption in [16]. However, there are still many open questions. Some of the most interesting questions include:

1. Can we improve the tolerance ratio for commitment and oblivious transfer, from $p \geq 2t + 1$ to say $p = 2t$?
2. Is there an efficient combiner for IND-CCA2 encryption, e.g. with no (or minimal) length increase?
3. Can we find (efficient) tolerant combiners for other tasks? For example, our early efforts for finding a tolerant combiner for $AONT$ [12] resulted in substantial loss in parameters (instead of s secret bits out of which l must remain secret, we need $2s$ secret bits out of which $s + l$ must remain secret).

Acknowledgment

Many thanks to Mihir Bellare, Ran Canetti, Shai Halevi, Kath Knobe, Yehuda Lindell, Boaz Patt-Shamir, Avi Wigderson and anonymous referees for helpful comments and discussions. This work was supported in part by Israeli Science Foundation grant ISF 298/03-10.5.

References

1. C. A. Asmuth and G. R. Blakley. An efficient algorithm for constructing a cryptosystem which is harder to break than two other cryptosystems. Comp. and Maths. with Appls., 7:447-450, 1981.
2. B. Aiello, M. Bellare, G. Di Crescenzo, and R. Venkatesan, Security amplification by combiner: the case of doubly-iterated, ideal ciphers, Proc. of CRYPTO 98.
3. Jee Hea An, Yevgeniy Dodis and Tal Rabin, On the Security of Joint Signature and Encryption, in Theory and Application of Cryptographic Techniques, pp. 83-107, 2002. Also in Advances in Cryptology - EUROCRYPT 2002, volume 2332 of Lecture Notes in Computer Science, pages 83-107. Springer-Verlag, 2002.
4. Ross Anderson, Roger Needham. Robustness Principles for Public Key Protocols. In Proceedings of Int'l. Conference on Advances in Cryptology (CRYPTO 95), Vol. 963 of Lecture Notes in Computer Science, pp. 236-247, Springer-Verlag, 1995.
5. Martin Abadi, Roger Needham. Prudent Engineering Practice for Cryptographic Protocols. IEEE Transactions on Software Engineering, 22, 1 (Jan.), 1996, pp. 6-15.
6. Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS#1. In Advances in Cryptology - CRYPTO '98, LNCS 1462, pages 1-12. Springer, 1998.

7. M.Bellare, A.Desai, E.Jokipii, P.Rogaway: A Concrete Security Treatment of Symmetric Encryption, Proceedings of the 38th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 394-403, 1997. Revised version at http://www-cse.ucsd.edu/users/mihir/papers/sym-enc.html.
8. Mihir Bellare, Joe Kilian and Phil Rogaway, The security of cipher block chaining, Journal of Computer and System Sciences, Vol. 61, No. 3, Dec 2000, pp. 362-399. Extended abstract in Advances in Cryptology - Crypto 94 Proceedings, Lecture Notes in Computer Science Vol. 839, Y. Desmedt ed, Springer-Verlag, 1994.
9. Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic paradigm. In T. Okamoto, editor, Asiacrypt 2000, volume 1976 of LNCS, pages 531-545. Springer-Verlag, Berlin Germany, Dec. 2000.
10. Mihir Bellare and Phillip Rogaway, Collision-Resistant Hashing: Towards Making UOWHFs Practical, Extended abstract was in Advances in Cryptology- Crypto 97 Proceedings, Lecture Notes in Computer Science Vol. 1294, B. Kaliski ed, Springer-Verlag, 1997. Full paper available at http://www.cs.ucsd.edu/users/mihir/papers/tcr-hash.html.
11. Joonsang Baek, Ron Steinfeld, and Yuliang Zheng. Formal proofs for the security of signcryption. In David Naccache and Pascal Pailler, editors, 5th International Workshop on Practice and Theory in Public Key Cryptosystems - PKC 2002, pp. 80-98, LNCS Vol. 2274, 2002.
12. Ran Canetti, Yevgeniy Dodis, Shai Halevi, Eyal Kushilevitz, Amit Sahai, Exposure-Resilient Functions and All-Or-Nothing Transforms, Advances in Cryptology - EUROCRYPT 2000. Lecture Notes in Computer Science, vol. 1807, Pages 453-469, Springer-Verlag, 2000.
13. Ran Canetti, Hugo Krawczyk and Jesper Nielsen, Relaxing Chosen-Ciphertext Security, Cryptology ePrint Archive, Report 2003/174. An extended abstract version appears in proceedings of CRYPTO '03, LNCS series, volume 2729, pp. 565-582, 2003.
14. Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography. In Proceedings of the 23rd Symposium on Theory of Computing, ACM STOC, 1991.
15. Ivan B. Damgård, Lars Ramkilde Knudsen. Enhancing the Strength of Conventional Cryptosystems, BRICS report RS-94-38, November 1994.
16. Yevgeniy Dodis and Jonathan Katz, Chosen Ciphertext Security of Multiple Encryption, in TCC 05, pp. 188-209, 2005.
17. Ivan B. Damgård, Torben P. Pedersen, Birgit Pfitzmann: On the Existence of Statistically Hiding Bit Commitment Schemes and Fail-Stop Signatures; Crypto '93, LNCS 773, Springer-Verlag, Berlin 1994, 250-265.
18. Ivan B. Damgård, Torben P. Pedersen, Birgit Pfitzmann: Statistical Secrecy and Multi-Bit Commitments; IEEE Transactions on Information Theory 44/3 (1998) 1143-1151.
19. Shimon Even and Oded Goldreich, On the Power of Cascade Ciphers, ACM Transactions on Computer Systems, Vol. 3, 1985, pp. 108-116.
20. National Institute of Standards and Technology, Federal Information Processing Standards Publication, FIPS Pub 180-1: Secure Hash Standard (SHA-1), April 17, (1995), 14 pages.
21. Oded Goldreich, The Foundations of Cryptography, Volume 1 (Basic Tools), ISBN 0-521-79172-3, Cambridge University Press, June 2001.
22. Oded Goldreich, The Foundations of Cryptography, Volume 2, ISBN 0-521-83084-2, Cambridge University Press, May 2004.

23. Oded Goldreich and Shafi Goldwasser and Silvio Micali "How to Construct Random Functions" Journal of the ACM, 33(4), 1984, 792-807.
24. Oded Goldreich, R. Impagliazzo, L. Levin, R. Venkatesen, D. Zuckerman. "Security preserving amplification of randomness", 31st Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, (1990), 318-326.
25. Shafi Goldwasser and Silvio Micali. "Probabilistic Encryption," JCSS (28), 1984, 270-299.
26. Philippe Golle, Markus Jakobsson, Ari Juels, and Paul Syverson. Universal reencryption for mixnets. In Tatsuaki Okamoto, editor, RSA Conference Cryptographers' Track, volume 2964 of LNCS, Springer-Verlag, pages 163–178, San Francisco, California, USA, February 2004.
27. Danny Harnik, Joe Kilian, Moni Naor, Omer Reingold and Alon Rosen: On Robust Combiners for Oblivious Transfer and Other Primitives. Proceedings of EUROCRYPT 2005, pp. 96-113, 2005.
28. Shai Halevi and Silvio Micali, Practical and Provably-Secure Commitment Schemes from Collision Free Hashing, in Advances in Cryptology - CRYPTO96, Lecture Notes in Computer Science 1109, Springer-Verlag, 1996, pp. 201-215.
29. Johan Hastad, Rudich Impagliazzo, Leonid A. Levin, and Mike Luby, Construction of a Pseudorandom Generator from any One-Way Function. SIAM Journal on Computing, Vol. 28, No. 4, pp. 1364-1396, 1999.
30. Amir Herzberg and Mike Luby, "Public Randomness in Cryptography", proceedings of CRYPTO 1992, ICSI technical report TR-92-068, October, 1992.
31. Amir Herzberg and Shlomit Pinter, "Composite Ciphers", EE Pub. no. 576, Dept of Electrical Engineering, Technion, Haifa, Israel, Feb. 1986.
32. Hugo Krawczyk, "The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?)," In Crypto '01, pp. 310-331, LNCS Vol. 2139, J. Kilian ed., Springer-Verlag, 2001.
33. Hugo Krawczyk and Tal Rabin. Chameleon signatures. In Network and Distributed System Security Symposium, pages 143-154. The Internet Society, 2000.
34. Arjen K. Lenstra and Eric R. Verheul. Selecting Cryptographic Key Sizes. Journal of Cryptology: The Journal of the International Association for Cryptologic Research, 14(4):255–293, September 2001.
35. U.M. Maurer and J.L. Massey, Cascade ciphers: the importance of being first, Journal of Cryptology, Vol. 6, No. 1, pp. 55-61, 1993.
36. Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone, Handbook of Applied Cryptography, Section 9.2.6, CRC Press, ISBN 0-8493-8523-7, October 1996. Available online at http://www.cacr.math.uwaterloo.ca/hac/.
37. Moni Naor and Moti Yung, Universal one-way hash functions and their cryptographic applications, Proc. 21st Annual ACM Symposium on Theory of Computing (STOC), 1989, pp. 33–43.
38. T. Dierks, C. Allen, The TLS Protocol: Version 1.0, Network Working Group, Internet Engineering Task Force (IETF). Available online at http://www.ietf.org/rfc/rfc2246.txt.
39. Eric Rescorla. SSL and TLS: Designing and Building Secure Systems. Addison-Wesley, 2000.
40. Adi Shamir, How to share a secret, Comm. of the ACM, 22(11):612-613, 1979.
41. Bruce Schneier, Applied Cryptography, John Wiley and Sons, 1996.
42. Victor Shoup, Using hash functions as a hedge against chosen ciphertext attacks, Adv. in Cryptology – Proc. of Eurocrypt '2000, LNCS 1807, pp. 275-288.

43. Adi Shamir and Yael Tauman. Improved online/online signature schemes. In Joe Killian, editor, Proceedings of Crypto 01, volume 2139 of LNCS, pages 355–367. Springer-Verlag, August 2001.
44. Yuliang Zheng, Digital signcryption or how to achieve cost(signature+encryption) << cost(signature)+cost(encryption), in Advances in Cryptology - CRYPTO'97, Berlin, New York, Tokyo, 1997, vol. 1294 of Lecture Notes in Computer Science, pp. 165–179, Springer-Verlag.
45. Phil R. Zimmerman. The Official PGP User's Guide. MIT Press, Boston, 1995.