Efficient Simultaneous Broadcast

Sebastian Faust[1], Emilia Käsper[1], and Stefan Lucks[2]

[1] K.U.Leuven ESAT-COSIC, Kasteelpark Arenberg 10,
3001 Leuven-Heverlee, Belgium
[2] Bauhaus-Universität Weimar,
99421 Weimar, Germany
{sebastian.faust,emilia.kasper}@esat.kuleuven.be,
stefan.lucks@medien.uni-weimar.de

Abstract. We present an efficient *simultaneous broadcast* protocol ν-SimCast that allows n players to announce independently chosen values, even if up to $t < \frac{n}{2}$ players are corrupt. Independence is guaranteed in the partially synchronous communication model, where communication is structured into rounds, while each round is asynchronous. The ν-SimCast protocol is more efficient than previous constructions. For repeated executions, we reduce the communication and computation complexity by a factor $\mathcal{O}(n)$. Combined with a deterministic extractor, ν-SimCast provides a particularly efficient solution for distributed coin-flipping. The protocol does not require any zero-knowledge proofs and is shown to be secure in the standard model under the Decisional Diffie Hellman assumption.

1 Introduction

1.1 The Simultaneous Broadcast Problem

Simultaneous broadcast allows n participants to simultaneously announce independently chosen values. It is useful in many applications such as auctions or coin-flipping, and is in fact a generic building block for any distributed protocol with an honest majority [16]. While this goal is trivial to achieve in a perfectly synchronous network where messages from all participants are broadcast at exactly the same moment, such a communication model itself is infeasible in practice. Instead, it is common to assume a partially synchronous network [6,14,15], where communication is divided into synchronized rounds, while every round is asynchronous, i.e., messages in a given round may arrive at any given moment within a time frame allocated to that round. Thus, in a partially synchronous network, every announced message may be chosen depending on all previously broadcast messages, including earlier messages received in the same round.

Consider the example of contract bidding where n players participating in a sealed bid auction wish to announce their bids in a "blind" way, such that the bids are revealed only once the auction is closed. In the partially synchronous model, simply announcing the messages in cleartext violates the requirement of blind bidding and allows the player speaking last to place the winning bid. At first sight, it seems sufficient to commit to a bid and only open the commitment after the bidding period has elapsed. However,

R. Cramer (Ed.): PKC 2008, LNCS 4939, pp. 180–196, 2008.

if Alice and Bob are competing players, then after seeing Alice's message, Bob may be able to create a related bid even if the commitment scheme is hiding. For example, Bob may simply copy Alice's message and thus guarantee that their bids are equal. In cryptography, such an (often undesirable) property is called *malleability* [9], and the attack is known as a *rushing attack*.

Secondly, it is often desirable that participants are bound to their commitments. If Alice and Bob use non-malleable commitments, Bob is not able to use the rushing attack to create a related bid.[1] He could, however, decide not to decommit at all after seeing Alice's bid, if the outcome is not to his favour. Thus, we need a simultaneous broadcast protocol that is both non-malleable—participants cannot choose their contribution based on other players' choices—and robust—nobody can pull out their contribution. Combined, this property is known as *independence*. Simultaneous broadcast protocols have many applications beyond contract bidding (see Sect. 4), and several solutions have been proposed to achieve independence in partially synchronous communication [6,14]. However, previous protocols require each party to broadcast $\mathcal{O}(n)$ messages and perform $\mathcal{O}(n^2)$ computation, so some authors use more efficient custom protocols for specific tasks such as coin-flipping [10]. In contrast, we propose a new *generic* simultaneous broadcast protocol that is particularly efficient in repeated runs.

1.2 Previous Work

The notion of non-malleability in cryptographic primitives was put forth by Dolev et al. [9]. In particular, non-malleable commitment schemes exhibit the property that, given a commitment $\mathsf{Com}(a)$, it is difficult to produce a commitment $\mathsf{Com}(b)$ to a related value b. More precisely, we require that if an adversary is capable of creating a commitment $\mathsf{Com}(b)$ satisfying some relation $R(a, b)$ then he is equally successful in creating such commitments without seeing $\mathsf{Com}(a)$ at all. Liskov et al. also introduced the notion of mutually independent commitments [19]; however, they propose a solution for the two-party setting, whereas we are interested in the multi-party case.

Recall that non-malleability alone does not provide independence since, after seeing honest players' values, malicious players may refuse to open their commitments. To ensure robustness in distributed computations, several authors have proposed to use verifiable secret sharing (VSS) to "back up" values. Rabin [21] and Gennaro et al. [15] propose to use *additive* (n-out-of-n) sharings of a joint secret key. Such an approach yields particularly efficient protocols for distributed signatures. For example, if an RSA signing key d is shared amongst n players as $d = d_1 + \cdots + d_n$, then each player's contribution to the signature $m^d \bmod N$ on message m is computed simply as $m^{d_i} \bmod N$. The novelty lies in the clever use of VSS to obtain robustness. Namely, they have every player verifiably share d_i of the key according to a (t, n)-threshold scheme. This assures that honest players can restore the contributions of failed players.

The idea of using VSS as back-up has since become quite well known. Returning to the case of commitments, the simple auction protocol can be made robust by having every participant VSS the committed value, as put forth by Gennaro [14]. We can thus enforce that all commitments are opened in the second round: if some player aborts, other

[1] The traditional notion of non-malleability does not, however, preclude exact copying of the commitment, so extra care must be taken to thwart the "copycat" attack.

players can open his commitment by reconstructing the shared value. Notice though that as opposed to the case of threshold signatures, Gennaro's broadcast protocol requires a new run of VSS for every round of broadcast, and additional zero-knowledge (ZK) proofs to ensure that the value under the non-malleable commitment is identical to the secret-shared value. We note that Pedersen's verifiable secret sharing [20] could also be used to provide simultaneous broadcast. This solution would eliminate ZK-proofs but not the communication overhead induced by verifiable secret sharing, and would be computationally heavier due to the use of Pedersen commitments.

1.3 Our Contribution

In many applications, the same set of parties need to perform multiple simultaneous broadcasts. For example, distributed statistical databases [10] require simultaneous broadcast for every database query. We present the first simultaneous broadcast protocol that significantly optimizes communication and computation cost for multiple invocations. Namely, in all previous solutions, verifiable secret sharing is required in every invocation of the protocol, even if the previous run was error-free. This means that each party has to broadcast more than t verification values and perform about tn exponentiations for verification. In contrast, we propose a new broadcast protocol ν-SimCast that requires one run of VSS in the initialization phase, after which multiple (ν) runs of broadcast can be carried out extremely efficiently. An error-free execution requires only two rounds, during which each party broadcasts only one ciphertext and its decryption. Consequently, computation cost drops by $t/2$, since each party now needs to compute only $2n$ exponentiations. For $t \approx n/2$, we have order n gain in both computation and communication. In particular, even though ν-SimCast is optimized for repeated execution, 1-SimCast (a single execution of the protocol with $\nu = 1$) is no less efficient than previous solutions. Table 1 (Section 3.5) compares the performance of simultaneous broadcast protocols.

Our protocol does not require any zero-knowledge proofs and is thus proven secure in the *standard model* (Thm. 1). This makes ν-SimCast suitable for coin-flipping, since players do not need common (known in advance) randomness for non-interactive ZK-proofs to produce common (unpredictable) randomness as protocol output. We achieve this by combining Gennaro's idea of using semantically secure encryption for commitment with Rabin's idea of backing up secret keys through VSS. Our protocol achieves independence of outputs (following the definition by Gennaro [14]) with a reduction to the semantic security of ElGamal. We note that ElGamal can be substituted with any other semantically secure encryption scheme under somewhat stronger assumptions (the common random string model, or trusted setup).

In Section 3.4, we argue that ν-SimCast allows participants to broadcast multiple announcements in parallel. In addition to the broadcast function, we show how ν-SimCast can be used to generate random values (Cor. 1 in Sect. 3.3). In Section 4, we discuss how to optimize ν-SimCast even further to efficiently obtain random *bits* rather than random group elements. These results provide a particularly efficient coin-flipping algorithm for e.g. the distributed databases example described above.

2 Preliminaries

2.1 Communication and Adversary Model

We consider a network of n players $\mathcal{P} = \{P_1, \ldots, P_n\}$. The players are pairwise connected through private point-to-point links and have access to a reliable public broadcast channel. Messages sent via this channel are reliably delivered to all participants, i.e. all parties receive the same message. The existence of reliable broadcast channels is a common assumption for cryptographic protocols [6,14].

Private point-to-point links can be simulated by using encryption on the public channel. If physical broadcast channels are not available, they can be implemented with special broadcast protocols [3,18]. However, reliable broadcast is costly when implemented on realistic networks such as the Internet. The protocol of Cachin et al. has message complexity $O(n^2)$ when run amongst a group of n parties and is "only" probabilistic, i.e., it introduces a small error probability.

In our setting we allow the adversary $\mathcal{A}$ to corrupt an arbitrary set of $t < n/2$ players. Corrupt players can act in any way during protocol execution, including protocol violation and early abort. The adversary is considered to be static, i.e., the set of corrupt players is fixed before the protocol execution. A special broadcast protocol further restricts the corruption tolerance to $t < n/3$, although it is possible to keep the resilience at $t < n/2$ under certain additional assumptions (e.g., the existence of a PKI) [18].

We structure the communication in rounds, and model delay in the transmission of messages by assuming partially synchronous communication. In contrast to the perfectly synchronous model where all messages in a given round are delivered simultaneously, the partially synchronous model allows an arbitrary delay within each round. In practice, such a model can be implemented by using synchronized clocks: if a participant does not finish its operations during a predefined time frame, he is disqualified from further processing. In a way, the partially synchronous communication model augments the adversary's power by allowing to fix the delay of messages sent by corrupt parties. As a consequence, a protocol that claims to be secure in the partially synchronous model has to withstand an adversary that speaks last in each round and incorporates all information learned from *all* honest parties in the same as well as previous rounds.

2.2 Cryptographic Components

In the following, we use the concept of a negligible function $\varepsilon(k)$ to express that for every constant $c \geq 0$ there exists an integer k_c such that $\varepsilon(k) < k^{-c}$ for all $k \geq k_c$.

Semantically Secure Encryption. We model public key encryption as a triple of probabilistic polynomial-time algorithms Gen, Enc and Dec for key generation, encryption and decryption, respectively. Intuitively, a public key encryption scheme is said to be semantically secure if a ciphertext does not reveal any information on the encrypted message other than what is known *a priori*. This is formalized as a game $\mathsf{Sem\text{-}Sec}$

where the adversary $\mathcal{A}$ has to guess a bit b corresponding to the correct plaintext. Let $\mathcal{R}$ be the appropriate domain of randomness:

$\mathsf{Sem\text{-}Sec}[k]$:
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$;
$(\mathsf{state}, m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk})$;
$b \leftarrow \{0,1\}; r \leftarrow \mathcal{R}$;
$c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_b, r)$;
output $\mathcal{A}(\mathsf{state}, c)$;

The semantic security of the scheme is then quantified by the adversary's success probability.

Definition 1. *A public-key encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is said to be* semantically secure *if for any probabilistic, polynomial-time bounded adversary* $\mathcal{A}$ *the advantage* $\varepsilon(k) = \Pr[\mathsf{Sem\text{-}Sec}[k] = b] - \frac{1}{2}$ *is negligible in the security parameter* k.

In our construction, we explicitly require that the encryption scheme is *committing*, i.e., no two different messages encrypt to the same ciphertext under the same public key.

ElGamal Encryption. Let p and $q|p-1$ be primes. Let $g \in \mathbb{Z}_p^*$ be the generator of a cyclic group G of prime order q. Recall that given a secret key $x \in \mathbb{Z}_q$ and the corresponding public key $y = g^x$, a (randomized) ElGamal encryption of a message $m \in G$ is a tuple $c = (g^r, y^r m)$, where $r \in \mathbb{Z}_q$ is chosen uniformly at random. The semantic security of the ElGamal scheme is equivalent to the Decisional Diffie-Hellman assumption [23]. ElGamal is a committing encryption scheme: given an ElGamal public key y, one can commit to a message m by $\mathsf{Com}(m) = c = (g^r, y^r m)$ and decommit by revealing (r, m). Naturally, the same commitment can also be opened by anyone who knows the secret exponent x. This property will be crucial for us in achieving robustness.

Verifiable Secret Sharing. In a (t, n)-threshold secret sharing scheme, a dealer D shares a secret s amongst a group of players $\mathcal{P} = \{P_1, \ldots, P_n\}$ during the Share phase by sending a share s_i to P_i. In the Recover phase, a group of at least $t+1$ players can reconstruct the secret s, using their shares s_i. Unfortunately, simple secret sharing suffers from two drawbacks: first, a corrupt dealer can easily distribute inconsistent shares. Second, other share-holders cannot detect a corrupt share-holder P_j presenting a fake share s'_j in the Recover-phase. A verifiable secret sharing scheme (VSS) solves both problems by adding a third primitive Verify that allows parties to verify the consistency of sharing and recovery. As an inherent property, VSS guarantees that if D is not disqualified during the sharing process, then any set of $t+1$ shares of honest parties define the same unique secret s (except with possibly a neglible error probability). Unless mentioned otherwise, we assume that the reconstruction error is zero.

Feldman VSS. Feldman's VSS scheme [11] builds on Shamir secret sharing [22] and consists of the following phases (omitting some details of error handling):

– Share: Let G be a cyclic subgroup of prime order q with generator g. To share a secret s, the dealer chooses a polynomial $f(x) = a_0 + a_1 x + \cdots + a_t x^t$, $a_{i>0} \in_R \mathbb{F}_q$

over the field $\mathbb{F}_q$ with $a_0 = s$ and degree t. The dealer sends each party P_i the share $s_i = f(i)$.
- Verify: The dealer broadcasts commitments $A_0 = g^{a_0}, A_1 = g^{a_1}, \ldots, A_t = g^{a_t}$ and each player P_i verifies $g^{s_i} \stackrel{?}{=} \prod_{j=0}^{t} (A_j)^{i^j}$.
- Recover: Given a set of $t+1$ shares $s_i = f(i)$, one can reconstruct the polynomial and find the secret free coefficient s by employing Lagrange interpolation. The validity of each submitted share can be verified as above.

In Feldman's VSS, a cheating dealer will always be caught. Finally, we will need the following result, stating that the scheme is perfectly simulatable:

Proposition 1. *Given any t shares of a secret s and the public value g^s, there exists an efficient simulator $\mathcal{S}$ that produces an outcome of the* Share *phase that is identical to the real execution of the* Share *phase.*

The simulation property shows that an adversary, controlling up to t participants, can compute consistent verification values A_i, $i = 1, \ldots, t$ himself. Thus, Feldman's VSS leaks no information about the secret beyond what is implied by the public value g^s.

Note that it is not know how to construct such a simulator for an adaptive adversary that may only corrupt some players at a later point. Thus, we present all security claims in the static adversary setting. In order for our protocol to achieve security against an adaptive adversary, one would first have to address the adaptive security of Feldman VSS [1].

Pedersen VSS. Compared to Feldman's VSS, Pedersen's scheme requires an additional element $h \in G$ (presumably generated by a trusted party during parameter setup) such that the discrete logarithm $\log_g h$ is kept secret. The sharing goes as follows:

- Share: To share a secret s, the dealer D now generates two degree t polynomials $f(x) = a_0 + a_1 x + \cdots + a_t x^t$ and $g(x) = b_0 + b_1 x + \cdots + b_t x^t$, where $a_0 = s$, and hands each participant two shares $s_i = f(i)$ and $s'_i = g(i)$.
- Verify: The dealer broadcasts commitments $A_i = g^{a_i} h^{b_i}$ for $i = 0, \ldots, t$. and each player P_i verifies $g^{s_i} h^{s'_i} \stackrel{?}{=} \prod_{j=0}^{t} (A_j)^{i^j}$.
- Recover: Given a set of $t+1$ shares $s_i = f(i)$, one can reconstruct the polynomial f and find the secret free coefficient s by employing Lagrange interpolation. The validity of each share can be verified as above, by having parties broadcast both shares s_i and s'_i.

Pedersen VSS assumes that a cheating dealer cannot solve the discrete logarithm problem. On the other hand, the next result shows that it guarantees unconditional privacy of the secret (while the privacy of Feldman's scheme is computational). More precisely, the adversary's view and thus actions are independent of the secret [20]:

Proposition 2. *For any (computationally unbounded) adversary $\mathcal{A}$ corrupting at most t parties and any view* $\mathsf{view}_{\mathcal{A}}$*,*

$$\Pr[D \textit{ has secret } s | \mathsf{view}_{\mathcal{A}}] = \Pr[D \textit{ has secret } s] \,.$$

3 The Simultaneous Broadcast Protocol ν-SimCast

3.1 The Basic Protocol

Our n-party protocol ν-SimCast allows each player P_i to announce a value u_i, such that the values announced by corrupt players are independent of the values announced by honest players. We divide the protocol into two phases: the Setup phase is executed only once, after which the SimCast phase can be iterated ν times sequentially or in parallel to announce ν values (where $\nu = \nu(k)$ is polynomial in the security parameter). The protocol has maximum possible fault tolerance: it remains secure if up to $t < n/2$ players are controlled by an adversary.

We first present a version of ν-SimCast using ElGamal encryption and Feldman's VSS. For simplicity, we also assume that all players use the same cyclic subgroup G of prime order q with generator g. In Section 3.2, we discuss other possible instantiations.

ν-SimCast$[t, n, G, g, k]$

I. Setup**:**

1. Share: Each party P_i generates an ElGamal key pair (x_i, y_i) and verifiably shares the secret key x_i using (t, n) Feldman-VSS. The public key $y_i = g^{x_i}$ is broadcast as a verification value during the Share phase.
2. Verify: Each party P_j verifies each share. If verification fails for some party P_i, P_j broadcasts a complaint against P_i.
3. For each complaint, P_i (as a dealer) reveals the correct share. Parties who receive more than t complaints or fail to deliver correct shares are disqualified. Each party builds the set of qualified parties $QUAL \subseteq P$.

II. SimCast (ν **iterations):**

Each party $P_i \in QUAL$ publishes an announcement u_i:

1. Encrypt: Each party $P_i \in QUAL$ wishing to announce u_i chooses a random value $r_i \leftarrow \mathbb{Z}_q$ and broadcasts a ciphertext

$$c_i = (g^{r_i}, y_i^{r_i} u_i)\,.$$

   If some party P_i does not broadcast a ciphertext, he is disqualified and his output is set to $u_i = \perp$.
2. Decrypt: For every published c_i, the party P_i broadcasts the decryption (u_i', r_i').
3. Recover: Each party P_j verifies the decryption values of each other party P_i by checking that $c_i \stackrel{?}{=} (g^{r_i'}, y_i^{r_i'} u_i')$. If verification fails for some P_i, parties run Recover to reconstruct the secret key x_i and compute the decryption $u_i = \mathsf{Dec}_{x_i}(c_i)$. Players who failed to deliver a valid decryption message are disqualified from the next iterations and the set $QUAL$ is updated.

Fig. 1. Simultaneous broadcast protocol ν-SimCast

Informally, the protocol works as follows. In the Setup phase, each player generates a key pair (x_i, y_i) for ElGamal and shares the secret key x_i amongst all players using (t, n) Feldman VSS. The SimCast phase consists of only two rounds of broadcast followed by fault handling:

1. Each player P_i broadcasts an ElGamal encryption $c_i = (g^{r_i}, y_i^{r_i} u_i)$, where $u_i \in G$ and $r_i \leftarrow \mathbb{Z}_q$ is the encryption randomizer;
2. Each player P_i reveals (u_i', r_i'). If the revealed values do not match, i.e., $c_i \neq (g^{r_i'}, y_i^{r_i'} u_i')$, players run the Recover phase of the VSS scheme to recover u_i.

Notice that it is also possible to decrypt the contribution of a corrupt player P_i without revealing his personal secret key x_i by using standard threshold decryption techniques. This may be useful if the adversary model includes *fail-corruptions* [12], where players are simply unavailable from time to time. As a drawback, ElGamal threshold decryption requires additional ZK-proofs to verify the validity of decryption shares.

For efficiency reasons, we may also allow parties not to contribute an announcement in an iteration of SimCast, as long as they faithfully participate in verification and reconstruction. Such a behavior can easily be integrated in our security analysis. Some applications such as coin-flipping do however require everyone to participate (see Cor. 1).

3.2 Generalizing ν-SimCast for other Cryptosystems

The instantiation of the ν-SimCast protocol using ElGamal encryption and Feldman VSS is particularly efficient: it does not require any zero-knowledge proofs and can be proven secure in the standard model. The fact that verifiably shared keys are never combined to a single threshold encryption/signing key allows us to use simple Feldman verifiable secret sharing in the Setup phase instead of the less efficient Pedersen VSS.

In principle, one could instantiate ν-SimCast, using any semantically secure committing encryption scheme and any suitable VSS scheme. However, the efficiency of ν-SimCast relies on the discrete-log setting in one intricate detail: we must ensure that the verifiably shared secret key indeed corresponds to the player's public key. Feldman VSS for ElGamal keys solves this problem automatically, since the public key g^{x_i} is broadcasted as a verification value during the Share phase and all players check that their received shares are consistent shares of the secret key x_i. This may require additional zero-knowledge proofs, and thus we may have to give up the standard model. Alternatively, one may assume trusted setup, which is a reasonable assumption in settings where malicious faults are expected to be relatively rare. Even under those assumptions, our scheme is likely to be more efficient than the previous protocol [14], which requires complex zero-knowledge proofs during every iteration (see Section 3.5 for details).

3.3 The Security of ν-SimCast

First, a secure simultaneous broadcast protocol should satisfy the basic properties of broadcast: the protocol outcome is *consistent* for all honest parties and each honest party *correctly* receives the announcement of each other honest party. In addition, we require

independence: for each iteration of SimCast, there should be no correlation between the announcements of corrupt parties and the announcements of honest parties.

Let $\mathcal{A}$ be a static polynomially bounded adversary that corrupts at most t out of the n parties and coordinates their action. Denote by $\mathcal{B}$ the subset of corrupt parties and set $\mathcal{G} = \mathcal{P} \backslash \mathcal{B}$. Consider one iteration of SimCast. Let $u_j \in G$ be the group element that P_j announces and let $u_{i,j} \in \mathcal{M} = G \cup \{\perp\}$ be the value that P_i receives as P_j's announcement. Set $\overrightarrow{U_i} = (u_{i,1}, \ldots, u_{i,n})$, i.e., $\overrightarrow{U_i}$ is the announcement vector received by P_i in one iteration of SimCast.

Our security definition of a simultaneous broadcast protocol is based on the definition introduced by Gennaro [14]. The latter requires that the output of any single corrupt party should be uncorrelated with the output of honest parties. Hevia and Micciancio [17] note that this definition does not capture the *collaboration* of corrupt parties, and bring an (admittedly artificial) example of a protocol that satisfies Gennaro's definition, but allows two corrupt parties to output values whose XOR is correlated to the output of honest parties. Thus, we modify the definition of independence to require that not only the output of a *single* corrupt party should be independent of the output of honest parties but also that there is no correlation between the announcement vector of *any* subset of corrupt and honest parties.

For each iteration of SimCast the following properties have to hold:

Consistency: For any $\mathcal{A}$, and for any pair of honest players P_i, P_j the probability $Pr[\overrightarrow{U}_i \neq \overrightarrow{U}_j]$ is negligible in the security parameter k.

Correctness: For any $\mathcal{A}$ and for any pair of honest players P_i, P_j the probability $Pr[u_{i,j} \neq u_j]$ is negligible in k.

Independence: For any $\mathcal{A}$, for any subset of corrupt players $Q \subseteq \mathcal{B}$, for all $\overrightarrow{m} \in \mathcal{M}^{|Q|}$ and all $\overrightarrow{u}, \overrightarrow{v} \in G^{n-t}$, we have that

$$|p^{Q}_{\overrightarrow{m},\overrightarrow{u}} - p^{Q}_{\overrightarrow{m},\overrightarrow{v}}| \leq \epsilon(k), \tag{1}$$

where $\overrightarrow{u}, \overrightarrow{v}$ are the announcements of honest players, ϵ is a negligible function of k and

$$p^{Q}_{\overrightarrow{m},\overrightarrow{u}} = \Pr[\text{Players in } Q \text{ announce } \overrightarrow{m} | \overrightarrow{u}]$$

denotes the probability that corrupt players in Q announce vector $\overrightarrow{m}$, given that honest players have announced $\overrightarrow{u}$.

Intuitively, the independence property of ν-SimCast follows from the fact that each player P_i *must know* the value u_i he chose to broadcast. Indeed, since P_i has verifiably shared his secret key x_i, he can always compute the decryption of the published value c_i. In approaches that combine non-malleable commitments with VSS-ing the value under commitment, complex ZK-proofs are required to ensure that the shared value is identical to the one under commitment. In contrast, knowledge of the secret key acts as an implicit proof of knowledge of the encrypted value and no additional proofs are required. We proceed to give a formal security proof.

Theorem 1. *Let $t < \frac{n}{2}$. If the Decisional Diffie-Hellman assumption holds in group G, then ν-SimCast$[t, n, G, g, k]$ is a simultaneous broadcast protocol.*

Proof. First, notice that in each iteration all honest parties use the same set $QUAL$, as disqualification of parties is done solely based on public information. In the following we set $\mathcal{B} = (\mathcal{P} \cap QUAL) \backslash \mathcal{G}$. It is easy to see that honest players are never disqualified.

Let $P_i, P_j \in \mathcal{G}$. If $P_\ell \in \mathcal{G}$, then $u_{i,\ell} = u_{j,\ell} = u_\ell$, since P_ℓ publishes the correct unique opening of c_ℓ. If $P_\ell \in \mathcal{B}$ then there are two options. First, P_ℓ does not broadcast c_ℓ. In this case $u_{i,\ell} = u_{j,\ell} = \perp$. Second, P_ℓ publishes a ciphertext c_ℓ but fails to decrypt it in Step 2. Since there are at least $t+1$ honest parties, the Recover-procedure of Feldman-VSS allows to reconstruct the unique value u_ℓ corresponding to c_ℓ, so $u_{i,\ell} = u_{j,\ell} = u_\ell$. This shows *consistency* and *correctness*.

The *independence* property is proven by reduction to the DDH assumption, or equivalently, the semantic security of ElGamal. Suppose that an adversary $\mathcal{A}$, given a security parameter k, achieves advantage $\varepsilon = \varepsilon(k)$. We build a second adversary $\mathcal{A}'$ that wins the semantic security game Sem-Sec$[k]$ with a related advantage ε', showing that $\varepsilon(k)$ must be negligible in k.

Assume that $\mathcal{A}$ corrupts t parties (wlog $\mathcal{B} = \{P_{n-t+1}, \ldots, P_n\}$) and that for at least one iteration $s \in [1, \nu]$ there exist two vectors $\overrightarrow{u}, \overrightarrow{v} \in G^{n-t}$, a subgroup $Q \subseteq \mathcal{B}$, and an announcement of corrupt parties $\overrightarrow{m} \in \mathcal{M}^{|Q|}$ such that $|p^Q_{\overrightarrow{m},\overrightarrow{u}} - p^Q_{\overrightarrow{m},\overrightarrow{v}}| > \varepsilon$ in iteration s. We use a similar hybrid argument as in [14]. Namely, for the vectors $\overrightarrow{u}$ and $\overrightarrow{v}$ in iteration s, define hybrids $\overrightarrow{u}^{(\ell)} = (v_1, \ldots, v_\ell, u_{\ell+1}, \ldots u_{n-t})$ for $\ell \in [0, n-t]$. Clearly, $\overrightarrow{u}^{(0)} = \overrightarrow{u}$ and $\overrightarrow{u}^{(n-t)} = \overrightarrow{v}$. Now,

$$\left| p^Q_{\overrightarrow{m},\overrightarrow{u}} - p^Q_{\overrightarrow{m},\overrightarrow{v}} \right| = \left| \sum_{\ell=1}^{n-t} (p^Q_{\overrightarrow{m},\overrightarrow{u}^{(\ell-1)}} - p^Q_{\overrightarrow{m},\overrightarrow{u}^{(\ell)}}) \right| \leq \sum_{\ell=1}^{n-t} \left| p^Q_{\overrightarrow{m},\overrightarrow{u}^{(\ell-1)}} - p^Q_{\overrightarrow{m},\overrightarrow{u}^{(\ell)}} \right| ,$$

so there must exist an index j for which

$$|p^Q_{\overrightarrow{m},\overrightarrow{u}^{(j-1)}} - p^Q_{\overrightarrow{m},\overrightarrow{u}^{(j)}}| > \frac{\varepsilon}{n-t}. \tag{2}$$

Wlog assume that $p^Q_{\overrightarrow{m},\overrightarrow{u}^{(j-1)}} - p^Q_{\overrightarrow{m},\overrightarrow{u}^{(j)}} > \frac{\varepsilon}{(n-t)}$ (otherwise we simply modify $\mathcal{A}'$ such that it flips the output of $\mathcal{A}$). Note that the hybrids $\overrightarrow{u}^{(j-1)}$ and $\overrightarrow{u}^{(j)}$ differ only in position j, where the corresponding values are u_j and v_j.

As specified in the game Sem-Sec, $\mathcal{A}'$ gets as input a challenge public key $\hat{y}$. We let $\mathcal{A}'$ choose $m_0 = u_j$ and $m_1 = v_j$ as the two messages. $\mathcal{A}'$ then obtains the challenge $c = \mathsf{Enc}_{\hat{y}}(m_b, r)$, where $b \leftarrow \{0,1\}$ and r is a random value. Now, $\mathcal{A}'$ runs $\mathcal{A}$. In the following, $\mathcal{A}'$ has to perform the steps of the protocol on behalf of the honest players $\mathcal{G}$ and simulate the view of $\mathcal{A}$:

1. For the simulation of the Setup phase, $\mathcal{A}'$ follows the protocol instructions for each player $P_i \in \mathcal{G} \backslash \{P_j\}$, i.e., he generates a key pair (x_i, y_i) and shares x_i. For P_j, $\mathcal{A}'$ deals t random shares to $\mathcal{A}$ and runs the simulator $\mathcal{S}$ from Proposition 1 on input $y_j = \hat{y}$ to publish the challenge public key $\hat{y}$ and appropriate verification values.
2. For iterations $1, \ldots, s-1, s+1, \ldots \nu$ of SimCast, $\mathcal{A}'$ simply follows protocol instructions. That is, for all honest players $P_i \in \mathcal{G}$, $\mathcal{A}'$ broadcasts a ciphertext c_i and its decryption.
3. For iteration s, $\mathcal{A}'$ follows the protocol instructions for all parties $P_i \in \mathcal{G} \backslash \{P_j\}$ using as announcement the appropriate value from the hybrid vector $\overrightarrow{u}^{(j)}$. For P_j, it publishes the challenge ciphertext c.

Since $\mathcal{A}'$ controls more than t parties, for all $P_i \in \mathcal{B}$ that have not been disqualified it has received $t+1$ shares of x_i in the Setup phase. This allows $\mathcal{A}'$ to decrypt P_i's encrypted announcements c_i and obtain u_i. Let $\overrightarrow{u}_Q$ be the announcements of the parties in Q. If $\overrightarrow{u}_Q = \overrightarrow{m}$ then A' outputs $b' = 0$; otherwise it outputs $b' = 1$.

First, we have to show that the simulation is indistinguishable from a real run of ν-SimCast.

Ad. 1: For all parties $P_i \in \mathcal{G}\backslash\{P_j\}$ our adversary $\mathcal{A}'$ follows exactly the protocol description. For P_j, $\mathcal{A}'$ uses the simulator $\mathcal{S}$ of Proposition 1 which produces a distribution that is identical to the distribution of a real execution.

Ad. 2: The simulation of iterations $1, \ldots, s-1, s+1, \ldots \nu(k)$ of SimCast is done as described in the protocol. Thus, both distributions are identical.

Ad. 3: $\mathcal{A}'$ simply follows the protocol, using announcements from hybrid $j-1$ (if $b = 0$) or hybrid j (if $b = 1$).

It remains to show that $\mathcal{A}'$ breaks the semantic security with a sufficiently large advantage ε':

$$\begin{aligned}\varepsilon' &= Pr[\text{Sem-Sec}[k] = b] - 1/2 = Pr[b' = b] - 1/2 \\ &= \frac{Pr[b' = 0 | b = 0] + Pr[b' = 1 | b = 1]}{2} - \frac{1}{2}\,.\end{aligned}$$

Notice that $\Pr[b' = 0 | b = 0] = p^{Q}_{\overrightarrow{m}, \overrightarrow{u}^{(j-1)}}$ and $\Pr[b' = 1 | b = 1] = 1 - p^{Q}_{\overrightarrow{m}, \overrightarrow{u}^{(j)}}$, so from above we get

$$\varepsilon' = \frac{p^{Q}_{\overrightarrow{m}, \overrightarrow{u}^{(j-1)}} + 1 - p^{Q}_{\overrightarrow{m}, \overrightarrow{u}^{(j)}}}{2} - \frac{1}{2} > \frac{\varepsilon}{2(n-t)}\,. \qquad \square$$

The following corollary shows that ν-SimCast can be used for fair coin-flipping. We discuss this application in detail in Section 4.

Corollary 1. *Let $\mathcal{A}$ corrupt at most $t < n/2$ parties. If ν-SimCast$[t, n, G, g, k]$ is used to announce values $u_i \leftarrow G$ chosen uniformly at random, then the product $u = \prod_{i=1}^{n} u_i$ is also random in G.*

Proof. The product $u = \prod_{i=1}^{n} u_i$ contains the random announcement u_j of at least one honest party P_j, which by Thm. 1 is independent from the announcements of corrupt parties. Thus, u is a random group element. $\square$

3.4 Parallel Execution of SimCast

Up to this point, we have considered the security of ν-SimCast in a strictly sequential communication model. This means that parties first execute the Setup phase and then sequentially execute ν iterations of SimCast. However, when our protocol is executed in a real-world network such as the Internet, it is often advantageous when instances of the protocol can be run in parallel. Unfortunately, parallel execution of protocols often makes the security analysis more subtle or even allows new attacks. Mostly, this is due to the need to rewind protocol execution in the simulation.

Table 1. Performance of simultaneous broadcast protocols with n participants and threshold t

	rounds	comm.	broad.	exponent.	rand. elem.	model	sec.
Gennaro-00 [14]	5	$\approx n+t+160$	$\approx t+160$	$\approx nt+160n$	$t+1$	CRS	DDH
Pedersen-VSS [20]	3	$2n+t+1$	$t+1$	$\approx nt$	$2t+1$	standard	DL
SimCast (setup)	2	$n+t$	$t+1$	$\approx nt$	t		
SimCast (iter)	**2**	**4**	**4**	**2n**	**1**	standard	DDH
1-SimCast	4	$n+t+4$	$t+5$	$\approx nt$	$t+1$		

Our protocol can be simulated without rewinding. Additionally, we do not require a full parallelization of ν-SimCast and rather focus on a simpler case where Setup is executed once after which the participants run iterations of SimCast in parallel, i.e. for all parallel instances, the Encrypt step of SimCast has to be completed before a single decryption takes place. Such a scenario is sufficient to decrease the running-time for many practical purposes (see Section 4). It is easy to see that the independence of non-decrypted announcements is still guaranteed, with a factor $1/\nu$ loss in the tightness of the reduction.

We believe that full concurrency of SimCast iterations is also possible but requires a more thorough analysis.

3.5 Performance Comparison for Simultaneous Broadcasts

We compare the performance of ν-SimCast with Gennaro's simultaneous broadcast protocol [14] and an approach based on Pedersen's verifiable secret sharing [20], which to the best of our knowledge are the most efficient solutions for simultaneous broadcast. For explicit comparison, we present all protocols in the same familiar discrete-log setting.

Table 1 summarizes the key properties. We count communication and computation cost in terms of group elements for a single player. For simplicity, we only consider exponentiations, as they dominate the computation cost. Additionally, we analyze the number of privately generated random group elements, the number of rounds and the number of broadcasts, as for practical implementations they are the most expensive factor.

All three protocols under comparison employ exactly the same mechanism—verifiable secret sharing—for error handling. Thus, we describe all protocols in the optimistic scenario, where all parties follow the protocol. Notice that since in the fault-free scenario no errors occur, no additional communication and computation is needed in the protocols' complaint phases. Also, in all our evaluations, we assume that polynomial evaluation does not require any exponentiations, i.e., that the values x^j are precomputed for all $x = 1, \ldots, n$ and $j = 0, \ldots, t$.

We start by briefly reviewing Gennaro's protocol, which we call Gennaro-00. The protocol consists of the following steps (note that we omit steps for verifying the zero-knowledge proofs):

1. Each party P_i publishes its own public key y_i.
2. P_i, wishing to announce u_i, publishes an ElGamal encryption $\mathsf{Enc}_{y_i}(u_i, r_i)$ and proves knowledge of u_i.

3. P_i verifiably shares u_i and proves in zero-knowledge that the VSS-ed value is identical to the encrypted value.
4. The parties process complaints.
5. Each P_i reveals the values u_i and r_i.

In the discrete-log setting, the proof in Step 2—knowledge of a value u_i encrypted under $y_i = g^{x_i}$ as $(g^{r_i}, y_i^{r_i} u_i)$—can be done efficiently by proving knowledge of the discrete logarithm of $\log_g y_i$.[2] The equivalence of the value under commitment and the value under VSS (Step 3) can be proven, using standard cut-and-choose techniques [2,4]. However, in order to guarantee that a cheating prover cannot succeed with probability greater than 2^{-n}, roughly n iterations are required. In other words, in order to achieve error probability 2^{-80}, the prover has to compute 80 ElGamal encryptions. Recently, Camenisch et al. proposed a practical verifiable encryption scheme that avoids cut-and-choose techniques altogether [5]. However, to guarantee soundness, the secret key of the encryption scheme has to be unknown to the prover. Thus, the scheme cannot be employed here, unless we assume trusted setup in Step 1.

To sum it up, Gennaro-00 runs in five rounds: in the first two rounds, each party publishes a public key, an ElGamal ciphertext and a (short) ZK-proof. Round 3 requires each party to privately send $n-1$ shares, and broadcast $t+1$ verification values for the polynomial together with a non-interactive ZK-proof involving 80 ElGamal ciphertexts. In Round 4 no extra work has to be done in the fault-free case. The last round adds two more broadcasted values. The total communication cost for one player is about $n+t+160$ group elements including the $t+3$ expensive reliable broadcasts. Computation cost is dominated by verification of shares and ZK-proofs—each party needs to compute about t exponentiations for each received share and 160 exponentiations for each proof, resulting in about $n(t+160)$ exponentiations for each player.

Second, we note that Pedersen's verifiable secret sharing (Pedersen-VSS) can also be employed for simultaneous broadcast. The security of the scheme follows from Proposition 2 and the hardness of the discrete logarithm (refer to [15] for a similar proof). It also requires an additional element $h \in G$ such that the discrete logarithm $log_g h$ is kept secret. Ignoring malicious faults, Pedersen-VSS then runs in three rounds, where in the first round each party P_i runs Share to announce a value $a_{i0} = u_i$, followed by a complaint phase and, finally, P_i opens the announcement by revealing a_{i0} and b_{i0}.

Compared to Gennaro's protocol, Pedersen-VSS does not require any zero-knowledge proofs and is thus also secure in the standard model. On the other hand, the VSS increases the amount of communication and computation, and each player needs to generate twice as many random elements for the coefficients of the polynomials. Both ν-SimCast and Gennaro-00 can employ the more efficient Feldman VSS scheme, even though standalone Feldman VSS is malleable [15].

The ν-SimCast protocol is comparable to Gennaro-00 and Pedersen-VSS in the setup phase, where verifiable secret sharing dominates the cost. However, each subsequent error-free iteration is much cheaper, requiring only 4 broadcast elements (one ElGamal ciphertext and its decryption from each player), $2n$ exponentiations for verifying the

[2] It is not guaranteed that each party actually knows the secret key corresponding to the public key y_i, and thus we indeed need an additional proof here.

decryption, and only a single random element for the ciphertext. To model the worst-case scenario when faults are frequent, we may look at the cost of 1-SimCast. We see that 1-SimCast still clearly outperforms Gennaro-00, and is slightly more efficient than Pedersen-VSS, at the cost of one extra round. However, in most applications that require simultaneous broadcast frequently, one does not expect malicious faults at every iteration, and thus ν-SimCast is clearly more practical than Pedersen-VSS. We discuss applications in detail in the next section.

4 Applications

The ν-SimCast protocol is a generic protocol that can be employed whenever players need to simultaneously announce independent values. As we have seen, this allows for the so-called sealed envelope auctions: non-malleability of SimCast guarantees that players cannot choose their bids to be higher than (or related in any other way to) previously announced bids; robustness further enforces that all "sealed" bids can later be opened.

Moreover, Corollary 1 shows that ν-SimCast can be used for joint generation of random values, opening up many applications beyond auction protocols. In particular, as our protocol does not employ zero-knowledge proofs, it can be used for the distributed generation of challenges for ZK-proofs without contradiction. We present some of the most prominent examples, and discuss efficiency matters.

4.1 Multi-Party Computation

The ν-SimCast protocol can be applied whenever a multi-party computation (MPC) protocol requires publicly known random values. As a prominent example, we present the Commitment Multiplication Protocol (CMP) [7,8] that is widely used in secure multi-party computation. Namely, in order to add verifiability to an MPC protocol and thus protect against active adversaries, players start by broadcasting commitments to their inputs. In order to detect malicious behaviour, each player then needs to create commitments to his output in a verifiable manner after every operation. Using a homomorphic commitment scheme, addition and multiplication with a public constant are straightforward operations: given a constant m, and P's commitments $\mathsf{Com}(a)$ and $\mathsf{Com}(b)$ to inputs a and b, everyone can compute commitments $\mathsf{Com}(a+b) = \mathsf{Com}(a) \cdot \mathsf{Com}(b)$ and $\mathsf{Com}(ma) = m\mathsf{Com}(a)$. Verifying the correctness of a commitment $\mathsf{Com}(c) \stackrel{?}{=} \mathsf{Com}(ab)$ is done interactively, using the following protocol:

1. P chooses a random β and broadcasts commitments $\mathsf{Com}(c)$, $\mathsf{Com}(\beta)$, $\mathsf{Com}(\beta b)$.
2. Other players jointly generate a random challenge r using 1-SimCast.
3. P opens commitment $\mathsf{Com}(ra+\beta)$ to reveal r' and commitment $\mathsf{Com}(r'b-\beta b-rc)$ to reveal 0.
4. Other players accept the commitment $\mathsf{Com}(c)$ iff all openings succeed.

Thus, such a protocol allows P to convince others that he has correctly generated a commitment to the product of two inputs without revealing anything about his inputs or output. More specifically, the protocol can be used to add verifiability to any MPC

protocol based on multiplicative secret sharing schemes (SSS). Namely, given shares of two secrets, any linear SSS allows participants to locally compute shares of their sum, and a multiplicative SSS allows to locally compute shares of their product. CMP then adds verifiability to the computations, since every participant can prove that he has correctly generated commitments to the new shares.

4.2 Coin Flipping

Our protocol can also be used in situations where random *bits* are required, rather than random group values. In practice, it is common to apply a hash function to the group element to obtain, say, a symmetric key from Diffie-Hellman key exchange. For a more rigorous approach, a recent result by Fouque et al. implies that in subgroups of $\mathbb{Z}_p^*$, efficient deterministic extractors exist in the standard model [13]. More precisely, the authors bound the distance from uniform of the k least significant bits of a random group element. For example, if p is a 2048-bit prime, then one can extract 128 bits with a bias $\varepsilon < 2^{-80}$ in a suitably sized prime order subgroup of Z_p^*.

Dwork et al. consider distributed noise generation for privacy-preserving statistical databases [10]. In order to guarantee a particular (Gaussian) distribution of the noise, their protocol requires n public random bits (where n is the number of participants). They obtain those bits by having each participant verifiably share out 2 bits, and then applying a deterministic extractor to the $2n$ low-quality bits to obtain n bits from a "close-to-uniform" distribution. Using 1-SimCast, we can directly obtain (a constant number of) random bits with a provably small bias in two rounds (excluding setup). Compared to the VSS-based solution, we again have a factor t gain. If one requires more random bits or stronger randomness guarantees than one execution of 1-SimCast can provide, we can run ν-SimCast with $\nu > 1$ *parallel* executions in two rounds.

5 Conclusion

ν-SimCast is an efficient protocol for simultaneous broadcasting that allows n parties to announce independently chosen values, even if up to $t < \frac{n}{2}$ players are corrupted. In contrast to previous solutions, our protocol only requires one run of verifiable secret sharing in the initialization phase, after which an arbitrary number of broadcasts can be carried out. During each broadcast, each party broadcasts only one ElGamal ciphertext and its opening, and verifies $n-1$ encryptions, which gives a factor $t \approx n$ improvement in communication and computation, compared to previous protocols. Also, our security properties do not rely on the usage of any ZK-proofs. Instead, we combine semantically secure encryption with backing up secret keys through VSS and obtain security in the standard model. Simultaneous broadcasting has various applications in distributed computations: for instance, ν-SimCast can be used to jointly generate random values. Multiple random bits can efficiently be extracted from the output of a single execution of 1-SimCast, making it practical in coin-flipping applications.

Acknowledgements. The authors thank Gregory Neven for many helpful comments. Emilia Käsper did part of this research while visiting the Computer Laboratory in the University of Cambridge, UK.

This work was supported in part by the Concerted Research Action (GOA) Ambiorics 2005/11 of the Flemish Government, by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT and the IAPP–Belgian State–Belgian Science Policy BCRYPT. Sebastian Faust is supported in part by a research grant of the IBBT (Interdisciplinary institute for BroadBand Technology) of the Flemish Government. Emilia Käsper is also partially supported by the FWO-Flanders project nr. G.0317.06 *Linear Codes and Cryptography*.

References

1. Abe, M., Fehr, S.: Adaptively secure Feldman VSS and applications to universally-composable threshold cryptography. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 317–334. Springer, Heidelberg (2004)
2. Asokan, N., Shoup, V., Waidner, M.: Optimistic fair exchange of digital signatures (extended abstract). In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 591–606. Springer, Heidelberg (1998)
3. Cachin, C., Kursawe, K., Petzold, F., Shoup, V.: Secure and efficient asynchronous broadcast protocols. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 524–541. Springer, Heidelberg (2001)
4. Camenisch, J., Damgård, I.: Verifiable encryption, group encryption, and their applications to separable group signatures and signature sharing schemes. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 331–345. Springer, Heidelberg (2000)
5. Camenisch, J., Shoup, V.: Practical verifiable encryption and decryption of discrete logarithms. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 126–144. Springer, Heidelberg (2003)
6. Chor, B., Rabin, M.O.: Achieving independence in logarithmic number of rounds. In: PODC 1987: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing, Vancouver, British Columbia, Canada, pp. 260–268. ACM Press, New York (1987)
7. Cramer, R., Damgård, I., Dziembowski, S., Hirt, M., Rabin, T.: Efficient multiparty computations secure against an adaptive adversary. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 311–326. Springer, Heidelberg (1999)
8. Cramer, R., Damgård, I., Maurer, U.M.: General secure multi-party computation from any linear secret-sharing scheme. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 316–334. Springer, Heidelberg (2000)
9. Dolev, D., Dwork, C., Naor, M.: Non-malleable cryptography (extended abstract). In: Annual ACM Symposium on Theory of Computing (STOC 1991), pp. 542–552 (1991)
10. Dwork, C., Kenthapadi, K., McSherry, F., Mironov, I., Naor, M.: Our data, ourselves: Privacy via distributed noise generation. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 486–503. Springer, Heidelberg (2006)
11. Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In: Proceedings of the 28th Annual Symposium on Foundations of Computer Science (FOCS 1987), pp. 427–437 (1987)
12. Fitzi, M., Hirt, M., Maurer, U.M.: Trading correctness for privacy in unconditional multiparty computation (extended abstract). In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 121–136. Springer, Heidelberg (1998)
13. Fouque, P.-A., Pointcheval, D., Stern, J., Zimmer, S.: Hardness of distinguishing the MSB or LSB of secret keys in Diffie-Hellman schemes. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 240–251. Springer, Heidelberg (2006)

14. Gennaro, R.: A protocol to achieve independence in constant rounds. IEEE Trans. Parallel Distrib. Syst. 11(7), 636–647 (2000)
15. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 295–310. Springer, Heidelberg (1999)
16. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Annual ACM Symposium on Theory of Computing (STOC 1987), pp. 218–229 (1987)
17. Hevia, A., Micciancio, D.: Simultaneous broadcast revisited. In: 24th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2005), pp. 324–333 (2005)
18. Katz, J., Koo, C.-Y.: On expected constant-round protocols for Byzantine agreement. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 445–462. Springer, Heidelberg (2006)
19. Liskov, M., Lysyanskaya, A., Micali, S., Reyzin, L., Smith, A.: Mutually independent commitments. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 385–401. Springer, Heidelberg (2001)
20. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992)
21. Rabin, T.: A simplified approach to threshold and proactive RSA. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 89–104. Springer, Heidelberg (1998)
22. Shamir, A.: How to share a secret. Commun. ACM 22(11), 612–613 (1979)
23. Tsiounis, Y., Yung, M.: On the security of ElGamal based encryption. In: Imai, H., Zheng, Y. (eds.) PKC 1998. LNCS, vol. 1431, pp. 117–134. Springer, Heidelberg (1998)