SOCIETY FOR SCIENCE AND EDUCATION
UNITED KINGDOM

TNC TRANSACTIONS ON NETWORKS AND COMMUNICATIONS

VOLUME 8, NO. 6
ISSN: 2054 -7420

Practical Generation of Common Random Strings for Secure Multiparty Computations Over the Internet

István Vajda
Technical University of Budapest, Department of Informatics, Budapest, Hungary
vajda@hit.bme.hu

ABSTRACT

It is known that most of the interesting multiparty cryptographic tasks cannot be implemented securely without trusted setup in a general concurrent network environment like the Internet. We need an appropriate trusted third party to solve this problem. An important trusted setup is a public random string shared by the parties. We present a practical n-bit coin toss protocol for provably secure implementation of such setup. Our idea is inviting external peers into the execution of the protocol to establish an honest majority among the parties. We guarantee security in the presence of an unconditional, static, malicious adversary. Additionally, we present an original practical idea of using live public radio broadcast channels for the generation of common physical random source.

Keywords: secure coin-tossing; cryptographic protocols; universal composition; source of entropy

1 Introduction and related works

When we run an instance of our protocol, in general, we run it in an environment where arbitrary number of instances of arbitrary (potentially harmful) protocols may run simultaneously. Such an environment is called a general concurrent network environment (a.k.a. Universal Composition, UC). Accordingly we want to provide guaranties of security when the protocol is used as a component of a larger system like the Internet. Most of the interesting multiparty cryptographic tasks cannot be implemented securely even in the plain model (when authenticated communication links are assured) if the target protocol instance runs in a general concurrent network environment (Canetti, 2001; Canetti and Fishlin 2001; Canetti et al., 2006). We need an appropriate trusted third party. For example, otherwise it would not be possible to securely realize important applications such as e-commerce, e-banking, e-voting, etc.

In the common reference string (CRS) model any functionality can be computed UC-securely, for any number of corrupted parties (Canetti at al., 2002). In the CRS model, we can achieve computational zero-knowledge without requiring interaction (Blum et al., 1988). The CRS is a generalization of the common random string model.

A coin-tossing protocol lets two parties decide on an n-bit string that should be a random (or at least pseudorandom) string. The classic two-party protocol of Blum for the remote tossing of a single bit assumes a perfectly binding commitment scheme (Goldreich, 2004, Section 7.4.3.1. Vol II.). Its generalization to the task of the generation of common random string (with constant round protocol) additionally assumes (constant round) zero-knowledge proof of knowledge for any NP-relation (Lindell,

DOI: 10.14738/tnc.86.9447
Publication Date: 21st th January 2021
URL: http://dx.doi.org/10.14738/tnc.86.9447

2003). Both protocols are secure in the presence of static, malicious adversaries and stand-alone setting, where security means that these protocols securely emulate their respected ideal functionalities that output common uniformly random bits and strings, respectively. Note, these protocols guarantee secure pseudorandom output.

Assuming constant round perfectly hiding commitment scheme and constant round perfect zero-knowledge arguments of knowledge, constructions for perfect constant round coin tossing protocols for the generation of common random string in the presence of static, malicious adversary were published in (Lindell, 2003; Zhang et al., 2010). Perfectness means that the *output* of the coin-tossing protocol is *statistically* close to *uniform,*

In all of these references, security is considered with abort. Recall, it is impossible for two parties to toss a coin fairly, so that neither party can cause a premature abort or bias the outcome (Cleve, 1986).

In this work, we present a protocol that outputs statistically uniform common n-bit string with overwhelming probability in the presence of an unconditional, static, malicious adversary. The informal idea underlying the construction of the unconditionally secure protocol is the following. Alice and Bob invite Cecil (an external party) to take part in a 3-party computation of a common random string. If Cecil is honest, then the honest majority is established. In the case of an honest majority, honest parties will receive common uniformly distributed random string. However, when Cecil is corrupted and collaborates with dishonest Alice (or Bob) it will result in an honest minority and the adversary can control the output. Alice and Bob will run the main 2-party protocol such that the protocol calls the 3-party protocol as its subroutine several times with different external parties and computes the final common random string by combining the elements of a set of outputs of 3-party instances (using simple XOR operation). If these XOR-ed n-bit strings are statistically independent then the final output becomes uniformly distributed assuming that at least one string in this set is computed with an honest majority. Accordingly, the task is to prevent the adversary from forcing outputs of the 3-party executions without an honest majority to distort the output of the 2-party protocol.

We also present an original practical idea for the implementation of such a random source. The trivial and cheap access to live public radio broadcast channels by parties located even within a large geographical area offers a good opportunity for a trustable raw source of common randomness. For processing the audio channel, we propose the use of the audio fingerprinting technique (Cano et al., 2002). In the original technique of audio fingerprinting, a "noisy" audio fingerprint is produced from the audio channel and compared to a stored perfect “etalon” produced from an MP3 record. The original intended application of such fingerprints is (automatic) identification songs without using external metadata or embedded watermarking. All receivers within the coverage of the transmission have to be able to extract essentially the same bit-string from the noisy received signal. In comparison, our task is not the identification of songs (or any piece from audio media, like a commercial) but a simultaneous generation of strings of bits with “high similarity” at different geographical locations by listening to the same live public radio channel.

The organization of the paper is as follows. In Section 2 we give our models and assumptions. In Section 3 we briefly summarize our contributions. We present our protocols, in Section 4 and Section 5. In Section 6 we propose a practical idea for the realization of a shared source of entropy, followed by a discussion.

2 Models and assumptions

Network setting: We assume a communication network with authenticated channels (plain model).

Adversary model: We assume unconditional, static, malicious corruption adversary. One of the parties (A or B) may be corrupted from the beginning of the run of the protocol.

Our model for the corruption of an external peer coming from the total population of parties is simple. Let p denote the probability that the adversary can corrupt an external party chosen arbitrarily from the population.

We consider adversaries with the following malicious behavior. We assume that the goal of the adversary is to force the value of the common string so that it becomes a sample randomly chosen from some distribution *D* different from the uniform and decided by the adversary before the beginning of a new execution of the protocol. Though such an assumption is a restriction on possible malicious strategies it is a meaningful assumption if we consider scenarios where the adversary can make a protocol vulnerable by choosing the distribution of common strings used in the protocol appropriately.

Shared resources: We will assume that common public time value *t* is available for the participants.

Security with abort/bias: We cannot prevent the adversary from causing a premature abort when the 3-party subroutine runs with no honest majority.

External party: For the simplification of the analysis, we assume that an external party never refuses an invitation to take part in running the 3-party subroutine.

We note the analysis can be extended in a straightforward way to include a probabilistic model for the willingness of an external party to accept an invitation.

3 Our contributions

We present a protocol for a computationally unconditional, static, malicious adversary. The protocol outputs uniformly distributed common random string essentially with probability at least $1 - 2^{-ms}$ integer *m* and *s* are adjustable parameters of the protocol. (Claim 1, Claim 2). In this construction, we propose the idea that Alice and Bob invite external peers to the process of generation of uniformly distributed common random strings.

We also present an original idea for the realization of a shared source of entropy. We point out that the trivial and cheap access to live public broadcast channels by parties located even within a large geographical area offers an interesting opportunity for a trustable raw source of common randomness.

4 The protocol

We present a coin-tossing protocol that is secure in the presence of a static, malicious adversary with unconditional complexity.

Protocol π consists of three parts:

- a 3-party subroutine
- a preprocessing algorithm for the pre-generation of a set of common random strings,
- an algorithm for the computation of the output.

Copyright © Society for Science and Education, United Kingdom

The main steps of the execution of the protocols are shown in Fig. 1.

Protocol π

Parties: A, B, external parties

Input parameters: integers *M*, *N, n*

1) Party A and B run the preprocessing algorithm:

Parties repeat *M*-times:

1.1) Parties execute the 3-party subroutine
1.2) Parties store the output string of the subroutine in their storage of strings

2) Parties run the main algorithm:

2.1) Parties run the index set algorithm on their own that outputs an index set

2.2) Parties compute the output on their own that is the XOR sum of the strings in the storage of strings selected by the index set

Figure 1. Main steps of the protocol

4.1 The 3-party subroutine

The execution of the 3-party subroutine consists of two steps (Fig.2). First, the parties choose an external party. Next, they run a 3-party coin-tossing protocol.

Let $\Sigma(\lambda,\lambda,\lambda)=(R^A,R^B,R^C),\ R^X\in\{0,1\}^n, X\in\{A,B,C\}$ denote a 3-party coin-tossing functionality such that

1) in the case of an honest majority, honest parties will receive common uniformly distributed random string R chosen from the uniform distribution over *n*-bit strings;
2) in the case of no honest majority the adversary can decide between the following options:
   - biasing the distribution: the output of an honest party is chosen by the adversary from an arbitrary distribution over *n*-bit strings,
   - aborting the execution: honest party outputs $\perp$,
   - acting honestly: parties output common *n*-bit string with uniform distribution.

Notice, in the case of honest majority the adversary is not able to prevent honest parties from getting a common uniformly distributed random string. In contrast, in the case of no honest majority, we allow for the adversary the full control over the output of an honest party.

Note, the adversary is always aware of the actual case about the majority. Indeed, by standard assumption, the adversary controls all dishonest parties.

We recall the following results for secure multiparty computation with an honest majority:

- every function can be securely computed with statistical security in the malicious model assuming authenticated broadcast channels are available (Rabin and Ben-Or, 1989),
- delivery of output can be guaranteed, i.e. honest parties always learn the output even if some parties abort prematurely,

- any functionality can even be UC-securely computed without external trusted setup (Canetti, 2001; Canetti et al., 2002). The corresponding classic stand-alone results were extended in (Canetti et al., 2002) to the general concurrent running environment. It was shown that the main fundamental building block for a UC-secure realization is ideal functionality $F_{com}^{1:M}$ (one-to-many commitment), furthermore that this functionality can be realized in case of honest majority in the plain model (with authenticated broadcast channels), using known VSS (Verifiable Secret Sharing) protocols.

Let σ denote a 3-party coin-tossing protocol that UC-securely realizes functionality Σ in the case of honest majority in the presence of an unconditional adversary. In the sequel, we will assume that such a secure realization is available.

Let J denote the set of public identifiers of available external parties. We assume that all sizes/dimensions are adjusted properly throughout the algorithm. Elements of set *J* are arranged into a list in lexicographical order and labeled by indexes $1,2,...,|J|$. By using a deterministic mapping *T*, the time values (represented as different integers) are mapped into the range of indexes $(1,2,...,|J|)$. The assumption about mapping *T* is that it maps as uniformly as possible. If the range of time values has a smaller size than $|J|$ then different time values will be mapped into different index values. Otherwise, the mapping uses modulo $|J|$ arithmetic. Let ID denote the set of identifiers of external parties that have already been selected. Initially, set ID is empty. One of the parties will play the role S (Sender), the other the role Rec (Receiver) (e.g. they can decide this issue by the lexicographical order of their public identifiers).

3-party subroutine

Parties run the following algorithm for the invitation of a new external party:

1.1 Party S computes the identifier (C) of the new external party:
Party S chooses the smallest appropriate time value $t'=t+i,\ i=0,1,2...$ such that $T(t')$ is not an element of ID, where *t* is the local time at party S. Let $C=T(t')$. Party S sends message $(select_id,\ t,\ t',\ C)$ to party Rec.

Upon receiving a *select_id* message party Rec verifies its validity. If it is verified as valid, party Rec sends an acknowledgment message $(select_ack,\ C)$. Otherwise, party Rec aborts.

1.2. Upon receiving message $(select_ack,\ C)$ party S invites party C via running invitation sub-protocol ρ. Next, party S sends message $(invited,\ C)$ to party Rec. If party Rec does receive a valid message, it aborts. Party A and B store pair (*i*, *C*).

2. Parties A, B and C execute 3-party coin-tossing protocol σ, where C is the identifier from pair (*i*, *C*).

Figure 2. The 3-party subroutine

Copyright © Society for Science and Education, United Kingdom

4.2 The preprocessing algorithm

Assume, parties have storage for storing *M* binary strings each with a length of *n* bits (pool of strings). The indexes of strings (pool-indexes) come from range $\{1, \dots, M\}$. Parties A and B execute the preprocessing algorithm with input (A, B, M, n) to fill the pool with *M* common random strings. The algorithm calls the 3-party subroutine *M*-times (each with newly selected external party and independent local random elements) to compute the common random strings. The strings are stored sequentially in the pool.

4.3 The index set algorithm

To prepare for the idea of our construction, first, we present a natural approach that turns out to be vulnerable to attacks.

Example 1: Assume we choose a deterministic algorithm for the computation of pool-index such that the input is publicly available and changes in time. Time is a natural choice for such input in some resolution. The output of the algorithm is a set of *m* pool indexes such that different input values are mapped into different index sets. Otherwise, the algorithm is arbitrary.

Note, even before the execution of the preprocessing algorithm the adversary might predict future index sets. During the execution of the preprocessing algorithm the adversary could use successful predictions of index sets in the following attack:

Recall, an index set selects *m* strings from the pool of strings. Assume the adversary has successfully predicted an index set. The adversary waits for the computation of the pool entry with an index equal to the largest index within the predicted index set. If this entry is computed without an honest majority, the adversary forces a common string with a value equal to the XOR sum of the $m - 1$ elements of the subset that have already been computed plus the random sample from distribution *D.*

For concreteness, assume the adversary successfully guesses the day of execution of the protocol. Assume that a time value is structured as (year/month/day/hour/minute/ second). As there are 24·3600=86400 seconds a day, we get probability ~10^{-5} for a successful blind guess for the start of the execution, which is not a negligibly small value. Note, in a practical implementation we cannot assume a finer time resolution (than seconds) as we need to keep the cost of time synchronism throughout the system at an acceptable level.

At first glance, it might seem that we have a “plausible” way for predicting the timing of future actions harder for the adversary. The idea is that parties randomly and independently delay the time of sending messages ready for transmission. Note, however, such a solution could provide only partial protection. This is because there arise two contradicting requirements. On one hand, the magnitude of delay values (and of the magnitude of the accumulated delays) should be polynomial (in parameter *n*) to keep the time complexity polynomial. On the other hand, this size should at least be super-polynomial to foil guessing attacks on timing by an efficient adversary.

We have to prevent the adversary from the successful prediction of future index sets. Note, in addition to prediction, there is another potential attack. An adversary might attempt to manipulate the calculation of indexes and cause them to point to a subset of stored strings such that the subset has previously been fabricated during the preprocessing phase.

The storage of common random strings is divided into two parts and it is filled sequentially in time in two consecutive phases of preprocessing. The second part will be used for the computation of indexes (index sets), the first one stores the strings the indexes choose from (selected sets). The protocol is given in Fig. 3. an illustration is shown in Fig. 4.

Preprocessing:

Parties *A* and *B* fill the first and the second part of the pool of strings with M_1 and M_2 common random strings, respectively, by calling the 3-party subroutine $M = M_1 + M_2$ times.

Computation of indexes:

Parties consider a binary matrix *I* of size $n\ x\ M_2$, where the rows are the strings from the second part of the pool. The column vectors of matrix *I* are turned into indexes as follows:

The computation of a single subset of indexes with size *m* is as follows:

a) Let string $(i_1, \dots, i_s)$ be the binary representation of an index pointing into the first part of the pool, i.e. $= log_2(M_1)$, furthermore $s \leq M_2$. The *j*-th index $(i_{j1}, \dots, i_{js})$ of the subset is formed from the bits of the *j*-th column of matrix *I*, i.e. $(i_{j1}, \dots, i_{js}) = (I_{j1}, \dots, I_{js}),\ \ j = 1, \dots, m$, where $m \leq n$.

b) The computation of $t,\ t \geq 1$ subsets of indexes is as follows:

The second part of the pool is filled with M_2 strings, where $M_2 \geq tm(s/n)$. Disjoint *s*-bit segments of the colums of matrix *I* are turned into indexes in arbitrary order.

The index sets are stored by the parties. If an index set collides with a past one it is dropped and the computation of index set is repeated.

Computation of the output:

The selected *m* strings are XOR-ed. Different subsets of indexes can be used to generate outputs in any time order.

Fig.3 The n-bit coin-tossing protocol

An illustration of index computation is shown in Fig. 4. Horizontal rectangles represent common random strings. Dotted rectangles represent strings controlled by the adversary. The binary column vector (index) formed within the second (upper) part of the pool selects the i-th string of the first (lower) part of the pool.

Copyright © Society for Science and Education, United Kingdom

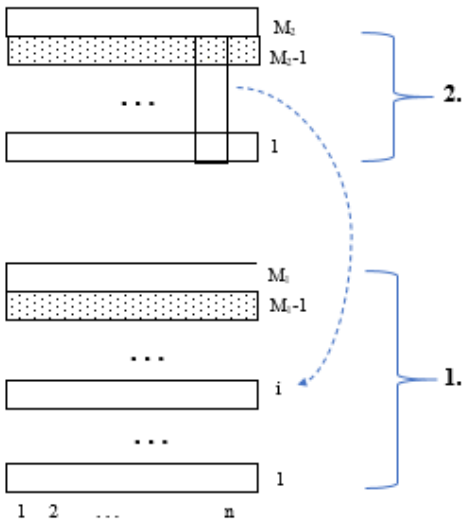


Fig. 4. Illustration of the computation of indexes

5 Analysis of the protocol

We prove the security of the protocol by using composition technique. First, we will consider the (stand-alone) security of the hybrid protocol π^{Σ}, where the subroutine runs ideal functionality Σ. We prove that the output of an honest party has uniform distribution except for an event with negligible probability.

Next, we replace the ideal calls $\Sigma_1, \dots, \Sigma_M$ with calls to their (unconditionally) secure realizations $\sigma_1, \dots, \sigma_M$, respectively. By our assumption, realization σ of ideal functionality Σ is UC-secure. Via the corresponding composition theorem, this implies that the security property of the hybrid protocol extends to the full-fledged real protocol.

The analysis is done under the assumption that no abort happens in the invitation phase. As we have no effect on the decision of the adversary on causing an abort event during the execution of the protocol, our claim about the output distribution of the protocol is conditioned on no abort event.

The formal definition of output distribution is the following. Once the protocol is called it can output N common random strings (at the request of the parties), denoted by a series $o_1, \dots, o_N$ of random samples. Random sample o_i, may be taken from a uniform distribution, from distribution D (wanted by the adversary) or from a general non-uniform distribution, $i = 1, \dots, N$. For instance, it may happen that in an actual run of the protocol, random sample o_1 is taken from a uniform distribution, random sample o_2 is taken from distribution D etc. For different calls of the protocol the series of these associated distributions may be different. The claims about the output distribution below provide upper bound on the probability that the distribution of random sample o_i comes from distribution D (Claim 1) or from a general non-uniform distribution (Claim 2), $i = 1, \dots, N$.

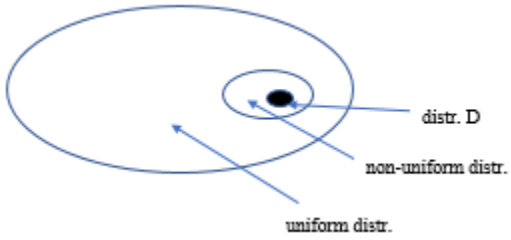


Fig. 5. The probability distribution of the output of the protocol

Fig. 5. illustrates the probability distribution of the output of the protocol. By Claim1 below the output can be a sample from a distribution *D* chosen by the adversary with negligible probability. It can happen that the attack is not successful (i.e. the output is not a sample from distribution *D*) however the attacking algorithm distorts the output distribution in an uncontrolled way, leading to some non-uniform distribution. We show that the output can have a non-uniform distribution with negligible probability, too.

Claim 1: The success probability of the assumed adversary against the hybrid 2-party coin-tossing protocol π^{Σ} is at most $\approx (NM_1p) \cdot 2^{-sm}$.

Proof: Let *h* denote the number of strings with uniform distribution computed with an honest majority in the first part of the pool (h-strings). Similarly, let *c* denote the number of strings controlled by the adversary (c-strings), where $h + c = M_1$.

Note, a c-string can be used to set the XOR sum to the wanted value for at most one subset of strings. Furthermore, the adversary cannot have control over the probability distribution of the XOR sum for further subsets of strings (i.e. in addition to this *c* subsets).

Let's consider a given prepared subset (of *m* strings from the first part of the pool). We compute an upper bound on the probability that an index set being computed will match this prepared subset. We prove that this probability is negligible. The main observation here is that the adversary cannot have any effect on the bits of an index string derived from strings that are computed with an honest majority (genuine bits). Besides, these bits are independent random bits with uniform distribution. In contrast, we conservatively assume that the complementary set of bits of indexes optimally favors the adversary (controlled bits). It means that we assume that each controlled bit of each string in the index set is equal to the bit of the index of a corresponding string from the prepared subset at the corresponding position (below we refer to such pairs of indexes as corresponding pairs).

It follows that probability γ that the two elements of a corresponding pair of strings match at all bit positions can be calculated as follows:

$$\gamma = \sum_{k=0}^{s} \left(\frac{1}{2}\right)^{k} (1-p)^{k} p^{s-k} = p^{s}\left(1 + \frac{q - q^{s+1}}{1-q}\right) \quad (1)$$

where $q = \frac{1-p}{2p}$. For small values of p formula (1) can well be approximated with exponent $\gamma \approx 2^{-s}$. Accordingly, the probability the index set selects the prepared set can be approximated by exponent $\gamma \approx 2^{-sm}$.

Since there are at most number c prepared subsets and at most N index sets are computed we get upper bound $Nc \cdot \gamma \approx Nc2^{-sm}$ on the probability of an event that there exists at least one matching pair conditioned on number c of strings. Recall, according to our corruption model, random variable c has binomial distribution $B(M_1, p)$. By knowing the probability distribution of variable c the success probability of the adversary can be bounded by

$$(NM_1p) \cdot \gamma \approx (NM_1p) \cdot 2^{-sm}$$

Example 2: For instance, with parameter values $M_1 = 32\ (s = 5), N = 32, p = 0.01, m = 16$ we get upper bound $\approx 2^{-77}$ for the probability of attack success.

Claim 2: The hybrid 2-party coin-tossing protocol π^{Σ} outputs non-uniform common random string with probability at most $(N\nu) \cdot 2^{-sm}$, $\nu \approx ((1-p)^{M_1+1})/(1-3p)$, $p < 1/3$ in the presence of the assumed adversary.

Proof: During an attack, the effect of c-strings can be visualized as follows:

Through the XOR operation, the randomizing effect of the h-strings is neutralized and then the distribution of the output string is set the wanted distribution. Different c strings neutralize different subsets of h-strings. Our point here is that the subset of c-strings can neutralize additional sets of h-strings within additional sets of strings of size m from the first part of the pool. And this latter effect is a side-effect of the attack algorithm, in the sense that it is not controlled by the adversary however it may lead to non-uniform ("blindly" distorted) output distribution.

The number of such c-string combinations can be bounded above by the number of all subsets, i.e. by power 2^c. Technically, we substitute variable c with 2^c in the proof of Claim 1. The expected value of 2^c can be bounded as follows

$$E(2^c) = \sum_{k=0}^{m} 2^k\, p^k (1-p)^{M_1-k} = (1-p)^{M_1}\left(1 + \frac{q - q^{m+1}}{1-q}\right)$$

$$\leq (1-p)^{M_1}\left(\frac{1}{1-q}\right) = \frac{(1-p)^{M_1+1}}{1-3p}$$

where $q = \frac{1-p}{2p}$.

For instance, for the parameters in Example 2 we get upper bound $\approx 2^{-76}$ on the probability that an output sample does not come from a uniform distribution.

In the next claim, we extend Claim 1 and Claim 2 to the full-fetched real protocol $\pi(\sigma_1, \dots, \sigma_M)$.

Claim 3: Claim 1 and Claim 2 remains valid for the composed protocol $\pi(\sigma_1, \dots, \sigma_M)$.

Proof: By our assumption, σ is a UC-secure of ideal functionality Σ realization. The claim is a straightforward consequence of the universal composition theorem.

Comments:

a) Recall, here we do not claim the UC-security, just the stand-alone security of the full fetched protocol π.
b) The protocol is secure under self-composition. Note, the honestly computed elements in different pools are statistically independent.
c) Practical observation is that the time complexity of the preprocessing phase can be decreased drastically by executing the 3-party subroutine in parallel instead of sequentially *M*-times.

5.1 Parameter setting

The parties can use the same pool of M_1 strings *N*-times to generate *N* common random strings. Appropriately choosing the parameters, we want to avoid the collision of outputs. Two selected sets may collide even if the corresponding index sets are different.

We want to choose parameters (M_1, N) such that the probability of such collisions of index sets becomes negligible. We consider the case which is the worst intuitively.

This is the case when all strings in the pool computed with honest minority are zero strings. In this case, collision means that the same subset of honestly computed strings (h-strings) is chosen at least twice. Let *h* and *z* denote the random variables of the number of h-strings and z-strings (zero strings) in the pool, respectively. According to our corruption model, random variable *h* has binomial distribution $B(M_1, 1-p)$.

Let $d(h)$ denote the number of different subsets of size *m* such that they lead to different output strings. Among such subsets, let $d(h,i)$ denote the number of those that contain number *i* h-strings, $0 \le i \le m$. For the cases, when $0 < i < m$ we get

$$\mathrm{d}(h,\mathrm{i}) = [h(h-1)\cdots(h-\mathrm{i}+1)] \,/\, \mathrm{i}! \ ,$$

if $h \ge \mathrm{i}$, else $d(h,i) = 0$. Note, $d(h,0) = 1$ (the case, when all elements are zero strings in a subset). Whence we get

$$\mathrm{d}(h) = \mathrm{d}(h,0) + \cdots + \mathrm{d}(h,\mathrm{m}).$$

Now, we compute the number of sets of size *N* (denoted by $D(h)$ where the elements are *m*-size subsets of strings leading to different outputs. We get

$$\mathrm{D}(h) = [\mathrm{d}(h)\cdot(\mathrm{d}(h)-1)\cdots(\mathrm{d}(h)-\mathrm{N}+1)] \,/\, \mathrm{N}!.$$

The ratio of the number of sets with *N* different outputs to the number of sets with *N* arbitrary outputs is

$$\mathrm{R}(h) = \mathrm{D}(h)/[\mathrm{d}(h)\mathrm{N}/\mathrm{N}!] \,,$$

whence

$$\mathrm{R}(h) \;=\; [1-1/\mathrm{d}(h)] \;\cdots\; [1 \;-\; (\mathrm{N}-1)/\mathrm{d}(h)].$$

Let $\mathrm{R} = \mathrm{E}[\mathrm{R}(h)]$ denote the average of random variable $\mathrm{R}(h)$ averaged over the binomial distribution $B(M_1, 1-p)$ of random variable *h*. Note, $1-\mathrm{R}(h)$ gives the ratio of output sets of size *N* with collisions in the outputs. Accordingly, $1-R$ is the average ratio of output sets with collisions. We choose parameters (M_1, N) such that parameter $1-R$ becomes negligible.

Example 3: Table 1 shows numerical results of the evaluation of formula for collision measure $1-R$ for different parameter settings, where the parameter of corruptibility is $p = 0.01$:

Table 1

N	M_1	*m*	*1-R*
20	25	12	3.6 E -5
30	30	15	2.8 E -6
40	40	20	5.6 E -9
50	50	25	9.6 E -12
60	60	30	1.5 E -14

We can make the following comments. The value of collision measure $1-R$ decreases with size M_1, roughly by two decades when we increase size M_1 by 10. The number of possible reusing of the pool is favorable, as it can be equal to the size of the pool. This fact outweighs the relatively large computational cost of preprocessing (the filling of the pool). Note, in this example parameter *p* was conservatively chosen.

The above numerical results suggest that the expected increase in computational time (because of such repeated computations) will be negligible even in case of lower M_1, N values (like those in the first row of Table 1).

5.2 The case of a stronger model of corruption

In this subchapter, we extend our model for the corruption of an external party. So far, we considered *Corruption_model_1*, where p denotes the probability that the adversary can corrupt an external party chosen arbitrarily from the population.

A stronger model is the following *Corruption_model_2*. By this model, the adversary can corrupt any subset of the population with size at most $100p$ percent of the population.

Note, these corruption models are similar in the property that the adversary can corrupt 100*p* percent of the population on average. In the case of the first model, the adversary can corrupt parties independently with probability *p*. In the second model, the adversary can corrupt any external party with an upper bound on the total number of corruptions. Nevertheless, *Corruption_model_1* can be realistic in scenarios where we can make the natural assumption that different members of the population installed protection of different strengths around their resources and so the success of the adversary may vary from party to party.

Recall, here we assume static corruption, i.e. the adversary has to decide about corruption before the execution of the protocol, in particular before the preprocessing stage starts. On the contrary, an adaptive

adversary under *Corruption_model_2* could easily break the security of the protocol in the case of $M \leq pQ$, where *Q* is the size of the population. Indeed, the adversary might corrupt all the external parties invited as a third party during the pre-processing phase.

6 A "practical" physical common random source

The trivial and cheap access to live public broadcast channels by parties located even within a large geographical area offers an interesting opportunity for a trustable raw source of common randomness. For processing the audio channel we propose an adaptation of the audio fingerprinting technique. For concreteness, we refer to audio fingerprints (Cano et al., 2002). Below we outline the idea.

In the technique of audio fingerprinting, a "noisy" audio fingerprint produced from the audio channel and compared to a stored perfect "etalon" produced from an MP3 record. The original intended application of such fingerprints is the identification of "songs" without using external metadata or embedded watermarking. In comparison, our task is not the identification of "songs" (or any piece from audio media, like a commercial) but "just" simultaneous generation of strings of bits with "high" similarity at different geographical locations by listening to the same live public radio channel. We face a few potential problems specific to our application. These are the following:

- attacks by a malicious party aiming to control the output sample,
- potential collisions of "fingerprint" samples generated by parties of running protocol instances concurrently in time,
- the potentially restricted size of the sample set.

Here we outline the ideas for solving these problems.

Malicious behavior:

Assume a 2-party protocol is running between Alice and Bob. Both parties generate their fingerprint, G_A , and G_B , resp. by listening to the same audio channel. If the parties are honest and the quality of the radio reception is good the fingerprint algorithm produces a highly similar pair of fingerprints ("similarity" is the term used in the original application (Cano et al., 2002)). If one of the parties is malicious and would like to replay a past fingerprint or manipulate its recent copy of fingerprint an honest party could reject it based on dissimilarity above a given limit. Parties keep only the (expectedly overwhelming) portion of their fingerprints identical in the two fingerprints.

The next problem is the selection of the common channel by the parties from the set of available ones:

Alice picks a source from the set of available sources and sends its index to Bob. In reply, Bob does the same. Alice and Bob generate a pair of fingerprints $\left(G_{1,A}, G_{2,A}\right)$ and $\left(G_{1,B}, G_{2,B}\right)$, resp. where $G_{i,C}$ is the fingerprint generated by party C from the i -th source, $i = 1,2$. Then they exchange their pairs of fingerprints and they carry out the processing mentioned above (similarity check and expurgation of dissimilarities) for the two left (*l*) as well as for the two right (*r*) elements of the pairs. The result is a pair $\left(G_l, G_r\right)$. Finally, the parties compute $G_l + G_r \pmod 2$ (in case of discrepancy in the length of strings the shorter length will be chosen).

Collisions of samples and the size of the sampling space:

A further problem arises with simultaneously running instances, by the potential collisions between fingerprints. Indeed, if (parties of) two different instances tune to the same audio source in an overlapping timeframe, they may derive the same CRS with non-negligible probability. We have to increase the number of simultaneously available sources to the maximum possible, i.e. make all live channels available by all potential parties: e.g. all radio broadcast channels within the hearing radius and/or all those accessible via the Internet. The advantage of the Internet is that it provides a tremendously large set of sources (from all around the Globe) with the disadvantage of potential oscillation in reception quality which might cause some delay (by retrials) in establishing a sound CRS. The advantage of traditional radio (e.g. a commercial FM channel) maybe its accessibility and high reception quality with the disadvantage of a substantially smaller size of accessible live sources (say restricted to 50 channels).

At the cost of complexity of processing each party could choose more than one ($k>1$) simultaneous sources (and combine the samples) to decrease the probability of CRS-collisions between concurrently running instances. For instance, with 1000 live Internet sources and $k=7$ the probability of collisions between pairs of CRSs drops below 2^{-55}.

Note, parties of an instance should not necessarily be in highly accurate time synchronism as for their time frames from within they generate their fingerprints. This is so because the fingerprint algorithms also have to take care of a similar problem in their original application scenario. In broadcast audio streams no markers are indicating the beginning and the end of the “songs” and matching algorithms provide a solution for the problem. Roughly, such algorithms look for matches within a longer extracted sequence to the stored etalon. In our task, we could rely on similar "synchronization", where parties exchange their longer sequences and search for matches.

REFERENCES

[1] Canetti, R. (2001) ‘Universally Composable Security: A New Paradigm for Cryptographic Protocols’, *In 42nd FOCS*, pp. 136-145

[2] Blum, M., Feldman, P. and Micali, S. (1988) ‘Non-Interactive Zero-Knowledge and Its Applications’, *Proceedings of the twentieth annual ACM symposium on Theory of computing (STOC 1988)*. pp.103–112

[3] Canetti, R., Lindell, Y., Ostrovsky R. and Sahai A. (2002) ‘Universally Composable Two-Party and Multi-Party Secure Computation’, *In Proc of the 34th STOC*, pp.494-503

[4] Canetti, R. and Fischlin, M. (2001) ’Universally Composable Commitments’, *Crypto 2001, LNCS, vol. 2139*, pp. 19-40

[5] Canetti, R., Kushilevitz E. and Lindell, Y. (2006) ’On the Limitations of Universal Composable Two-Party Computation Without Set-Up Assumptions’, *Journal of Cryptology,* Vol.19 No. 2, pp. 135-167

[6] Canetti, R., Pass, R. and Shelat, A. (2007) ‘Cryptography from Sunspots: How to Use an Imperfect Reference String’. *In FOCS '07* , pp. 249-259

[7] Cano, P., Batlle E., Mayer, H. and Neuschmied H. (2002) ‘Robust sound modeling for song detection in broadcast audio’, *Proc. 112th Audio Engineering Society Convention (AES)*

[8] Cleve R. (1986) 'Limits on the security of coin ips when half the processors are faulty', *In Proc. of the 18th STOC*, pp. 364-369

[9] Damgaard I. (2000) 'Efficient Concurrent Zero-Knowledge in the Auxiliary String Model'. *In Proc. of EUROCRYPT 2000*, pp. 418-430

[10] Goldreich, O (2004) 'Foundations of Cryptography Vol. I. - Basic Tools', *Cambridge University Press*, 2004.

[11] Goldreich, O (2004) 'Foundations of Cryptography Vol. II - Basic Applications', *Cambridge University Press*, 2004.

[12] Lindell, Y. (2003) 'General Composition and Universal Composability in Secure Multiparty Computation', *In Proc. of the 44th FOCS*. pp. 394–403

[13] Lindell, Y. (2003) 'Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation', *Journal of Cryptology*, Vol.16 No.3, pp 143–184.

[14] Rabin, T. and Ben-Or, M. (1989) 'Verifiable secret sharing and multiparty protocols with honest majority', *In Proc. of the 21st STOC*, pp. 73-85

[15] Zhang X., Hongxiang S., Zhang H., Qiaoyan W. and Haixia X. (2010) 'A Zhang X., Hongxiang S., Zhang H., Qiaoyan W. and Haixia X. (2010) 'A constant-round perfect parallel coin-tossing protocol', *In Proc. of the 2010 IEEE International Conference on Progress in Informatics and Computing*, Vol. 1, pp. 423-426

Copyright © Society for Science and Education, United Kingdom