Global, Unpredictable Bit Generation Without Broadcast

Donald Beaver [*] Nicol So [†]
Penn State University Penn State University

Abstract

We investigate the problem of generating a global, unpredictable coin in a distributed system. A fast, efficient solution is of fundamental importance to distributed protocols, especially those that rely on broadcast channels. We present two unpredictable bit generators, based on the Blum-Blum-Shub generator, that can be evaluated non-interactively; that is, each bit (or group of bits) requires each processor merely to send one message to the other processors, without requiring a broadcast or Byzantine Agreement.

The unpredictability of our generators (and the security of our protocols) are based provably on the QRA or the intractability of factoring. Remarkably, their structure seems to violate an impossibility result of [8], but our generators escape that lower bound because they achieve a slightly weaker goal: producing unpredictable bits directly, rather than producing "shares" of random bits. In doing so, they avoid the extra machinery (eg., "sharing shares") of similar results discovered independently in [8].

1 Introduction

Randomness has a variety of purposes in cryptography and computer science:

- **avoiding exhaustive search**: *eg.* finding a witness that a number is composite;
- **circumventing worst-case analysis**: *eg.* choosing a random pivot in Quicksort;
- **breaking symmetries**: *eg.* choosing a leader in a ring of processors;
- **hiding information**: *eg.* one-time pads;
- **measuring information**: *eg.* indistinguishability and Turing-like tests;
- **unpredictability**: *eg.* defeating an adversary's committed attack through unpredictable future choices.

For these and other reasons, a great deal of attention has been focused on ways to expand short (and scarce) random strings into long pseudorandom sequences.

For many applications, such as Byzantine Agreement [13, 4, 16] – the problem of agreeing on a common value in a network with unreliable nodes – randomness itself is simultaneously not enough and too much. Secret sharing, multiparty protocols, reliable decentralized databases, multicasts, and even timestamping protocols can depend very strongly on agreed-

[*]317 Pond Laboratory, Penn State University, University Park, PA 16802; (814) 863-0147; beaver@cs.psu.edu.

[†]Computer Science Dept., Pond Laboratory, Penn State University, University Park, PA 16802; so@cs.psu.edu.

upon information in a system. Thus, in addition to making efficient use of scarce random sources, we must also be worried about agreeing about the bits and protecting them from manipulation.

On the other hand, for Byzantine Agreement, the backbone of decentralized network protocols, it is not *randomness* but merely *unpredictability* that suffices to prevent an adversary's attack from having more than 50-50 chance of success. There are ingenious solutions [4, 16] requiring only two expected rounds of communication if a common but unpredictable coin is available.

In this paper, we show how to generate unpredictable bits efficiently and in one round of communication. In fact, this one round of communication is very simple: it requires each processor to send the same message to all the other processors, but it does not require the system to check whether each processor did so. We call this weak form of broadcast "*dissemination*," and we show how to achieve an unpredictable coin using one round of dissemination.

We present two solutions, one based on the QRA[1] and one based on the intractability of factoring. (The first solution is more efficient, but basing it on the weaker assumption (factoring) is not an obvious task.) Each solution uses a homomorphic technique for secret sharing in the style of Feldman [9]. (Caution is required, however; applying Feldman's VSS scheme as a "subroutine" leads to an easily predictable generator.) In a nutshell, it applies a generalized form of the Blum-Blum-Shub (BBS) pseudorandom number generator [7] to a hidden seed, and reveals the bits in a coordinated manner: no bit is revealed until some *$n - t$ processors decide it is time to do so, according to their programs.*

In [8], Cerecedo, Matsumoto and Imai have independently developed a method to generate secretly-shared bits using "no interaction." (Informally, a value is "secretly shared" if any collection of $n - t$ processors can determine it while any t or fewer cannot, where t is a bound on faulty processors.) Their method seems related to ours, but it requires more expensive computation to generate and verify the shares of the bits, as well as an order of magnitude increase in message size.

Furthermore, because our construction is based on the BBS generator, it is of a type that an impossibility result in their work excludes. It escapes impossibility partly because it attempts to achieve a weaker goal (obtain bits, not "shares of bits"). This weaker goal is sufficient to provide a very efficient implementation of Byzantine Agreement as well as any other protocol that relies on common, unpredictable bits to overcome an adversary.

2 Unpredictable Random Number Generators

This section provides background and technical lemmas for the main results described in §4.

We depart somewhat from the standard treatment of so-called cryptographically-strong pseudorandom number generators, in order to consider situations in which more than just the number stream itself becomes public. In particular, when a seed is revealed, it is clear that a pseudorandom bit stream no longer is indistinguishable from a uniformly random bit stream. It might be useful to permit a seed to be revealed at some later point (although, from the point of cryptographers bent on protecting communications, this seems to have no immediate use). In this case, we can no longer employ a standard treatment of pseudorandomness, but must turn to something new.

[1]The Quadratic Residuosity Assumption (QRA) states that no poly-size circuit family can determine whether x is a square modulo n with probability greater than $1/2 + k^{-c}$ - asymptotically, for any $c > 0$, for randomly chosen x with Jacobi symbol $+1$, and for n a randomly chosen Blum integer of size k.

The key property that we intend to capture is that the bits are unpredictable – namely, an adversary has little better than a 50-50 chance to predict the next bit, although once it is generated, the sequence may be clearly distinguishable from uniformly random bits because other (useless) information is present.

The Blum-Blum-Shub [7] generator provides a nice example. Briefly, it is based on selecting a random residue x modulo a Blum integer n, repeatedly squaring x, and outputting the least-significant bit of each successive value in reverse order $(\textsc{lsb}(x^{2^B}),\ldots,\textsc{lsb}(x^{2^1}))$. The bits themselves form a sequence indistinguishable from uniformly random bits [7, 1], but if the powers of x are revealed, even at some later point when the bits can be "permissibly" compromised, the whole view is clearly identifiable as the output of a generator (most truly random sequences would have no corresponding x and n which give rise to them).

This approach to randomness is suitable for problems such as Byzantine Agreement which require coin flips that are unpredictable, but the machinery used for generating the coins can be revealed afterward without harm. That is, the bits are used for algorithmic purposes rather than privacy. They should have the properties of a random sequence (50-50 chance of 0 or 1), and they should not be predictable, but they may be distinctly identifiable *after the fact* as the output of a generator.

Definition 1 *An* **Unpredictable Random Number generator (URN)** *G is a triple $(\hat{G},b,B)$, where $\hat{G}$ and b are poly-time algorithms and $B(k)$ is a polynomial, and such that on input k, integer i between 1 and $B(k)$, and a string x of length k, $\hat{G}$ outputs a string $\hat{G}(k,i,x)$, and $b(\hat{G}(k,i,x))$ outputs a bit. For initialization purposes, G may also output a string $\hat{G}(k,0,x)$.*

We call $b(\hat{G}(k,i,x))$ the i^{th} bit of G and we also write it as $G(k,i,x)$.

We define a poly-strong ("cryptographically-strong") URN as follows. Let P be a predictor, namely a poly-size circuit family that takes an input i and a string v_i (representing the output of the generator at the i^{th} step and the "revealable" information, *eg.* successive squares) and outputs a bit p_{i+1}. Let D be a distinguisher, namely a family of circuits that outputs 0 or 1.

Briefly, P attempts to predict the next bit at each stage; its guesses are compared to the answers, and the grades it gets ("right" *vs.* "wrong," as opposed to the answers themselves) should be pseudorandom.

Definition 2 *The* **prediction pattern** *of predictor P with respect to an URN G is the sequence of bits obtained as follows. Run P on $\hat{G}(k,0,x)$, $\hat{G}(k,0,x)\circ\hat{G}(k,1,x)$, $\hat{G}(k,0,x)\circ\hat{G}(k,1,x)\circ\hat{G}(k,2,x)$,* etc., *obtaining the* **predicted bits** *$a_1,a_2,\ldots,a_B$; The prediction pattern is the sequence:*

$$(P,G)(k)=(p_1,p_2,\ldots,p_{B(k)})=(a_1\oplus G(k,1,x),a_2\oplus G(k,2,x),\ldots,a_{B(k)}\oplus G(k,B(k),x)).$$

Let $\textsc{Unif}$ denote the uniform distribution on $\{0,1\}$.

Definition 3 *An URN is* **poly-strong** *if for all poly-size predictors P and distinguishers D, for all $c>0$,*

$$\left|D((P,G)(k))-D(\textsc{Unif}^{B(k)})\right|<\frac{1}{k^c},$$

where $D(Z(k))$ indicates the probability that D outputs 0 on strings sampled according to distribution $Z(k)$.

In the sequel, we often omit (k) for the sake of clarity.

An URN is stronger than a CSPRG, as is easily seen in the following:

Theorem 1 *If* $(\hat{G}, b, B)$ *is a poly-strong URN, then the bits output by* G *(ie.* $G(k,1,x), \ldots, G(k,B,x)$*, not the strings given by* $\hat{G}$*) form a cryptographically-strong pseudorandom sequence.*

Proof. If this sequence of bits fails a next-bit test, then the next-bit test can be used directly as an equally successful predictor against $(\hat{G}, b, B)$: given $\hat{G}(k,0,x), \ldots, \hat{G}(k,i,x)$, simply feed $b(\hat{G}(k,1,x)), \ldots, b(\hat{G}(k,i,x))$ to the next-bit test. □

2.1 A Poly-Strong URN Based on Blum-Blum-Shub

For purposes that will be clear in §4, we show that a generalized form of the BBS random bit generator provides an unpredictable number generator of a form suitable for distributed evaluation. Rather than repeatedly squaring, we repeatedly raise to a power that is itself a fixed power of two. Roughly speaking, we start with a slightly modified seed and follow the BBS approach, skipping over many of the bits that the BBS generator gives. We output these bits in reverse order.

Specifically, given $B(k)$, k, an integer n chosen randomly from BLUM_k, and poly-time computable functions $\alpha(k)$ and $\beta(k)$ with $\beta(k)$ odd, we use arithmetic over $\mathbf{Z}_n^*$ to define the URN $G_{\alpha,\beta}$ as follows (omitting k's for clarity):

$$\begin{aligned} \hat{G}_{\alpha,\beta}(k,i,x) &= x^{2^{(B-i)\alpha}} \\ b(y) &= \text{LSB}(y^{2^{\alpha-1}\beta}) \\ &\text{summarized by} \\ G_{\alpha,\beta}(k,i,x) &= \text{LSB}(x^{2^{(B-i)\alpha+\alpha-1}\beta}). \end{aligned}$$

The BBS generator is the special case of $\alpha(k) = \beta(k) = 1$. Like the BBS generator, $G_{\alpha,\beta}$ provides a poly-strong pseudorandom sequence of bits, under the QRA. Our unsurprising but necessary goal in this section is to show that, even knowing the bits and $x^{2^{(B-i)\alpha}}$ values used so far, a predictor cannot predict the next bit to come.

Lemma 2 $G_{\alpha,\beta}$ *is a poly-strong URN unless the QRA fails.*

Proof. We first show that a successful predictor would provide a way to compute $\text{LSB}(z^\beta)$ from z^2 (compare computing $\text{LSB}(z)$ from z^2 in the BBS case). Next, we show how to determine quadratic residuosity using the ability to compute $\text{LSB}(z^\beta)$ from z^2.

First, however, assume by way of contradiction that $G_{\alpha,\beta}$ is not a crypto-strong URN. Then there exists a $c > 0$, a distinguisher D, and a predictor P such that

$$\left|\Pr\left[D((P, G_{\alpha,\beta})(k)) = 0\right] - \Pr\left[D(\text{UNIF}^{B(k)}) = 0\right]\right| \geq \frac{1}{k^c}$$

for infinitely-many k. We omit the absolute value symbols without loss of generality.

Define a hybrid distribution by taking i bits of $(P, G_{\alpha,\beta})$, concatenating $B - i$ uniformly random bits to the end, and then running D on the result. The probability that D outputs 0 on the result is denoted by:

$$D_i = \Pr\left[D((P, G_{\alpha,\beta})[1..i] \circ \text{UNIF}^{B-i}) = 0\right].$$

Now, given some y, we wish to compute $\text{LSB}(z^\beta)$ where $z^2 = y$ and z is the principal square root of y. On input y, choose i from $\{1, \ldots, B\}$ at random, and run P on

$$y^{2^{(i-1)\alpha}}, y^{2^{(i-2)\alpha}}, \ldots, y.$$

P will output i guesses, $a_1, \ldots, a_i$; let $p_1, \ldots, p_{i-1}$ be the exclusive-or's with the known "correct" values (that is, $p_j = a_j \oplus \text{LSB}(y^{2^{(i-1-j)\alpha}\beta})$). Now, choose bits $p_i, \ldots, p_B$ uniformly at random. Run D on $(p_1, \ldots, p_B)$, and output $D \oplus p_i \oplus a_i$. (Intuitively, a 0 from D is more likely with distribution D_i than with D_{i-1}, which indicates that $p_i = a_i \oplus \text{LSB}(z^\beta)$.) Through a fairly standard argument, the probability that this method gives the correct answer for $\text{LSB}(z^\beta)$ is at least

$$\Pr[\text{correct}] \geq \sum_{i=1}^{B} \frac{1}{B} \cdot (\frac{1}{2} + D_i - D_{i-1} - o(\frac{1}{k^c}))$$

In other words, for infinitely many k and for some $c_1 > 0$, we have:

$$\Pr[\text{correct}] \geq \frac{1}{2} + D_B - D_0 - o(\frac{1}{k^c}) \geq \frac{1}{2} + \frac{1}{k^{c_1}}.$$

Finally, we must show how to turn this non-negligible advantage into a method to determine residuosity. Following [7], on input x, set $z = x^2$ and let w denote the square root of z that is a quadratic residue. Thus, $x = \pm w$ (else x has Jacobi symbol -1, and clearly is not a quadratic residue). Calculate $\text{LSB}(x^\beta)$ and compare it to the guess for $\text{LSB}(w^\beta)$ obtained by running the algorithm described above on z. If the bits are the same, then output "$x \in \text{QR}_n$," else output "$x \notin \text{QR}_n$." Observing that $x = -w \Leftrightarrow x^\beta = -w^\beta$ because β is odd, and that $\text{LSB}(-w^\beta) \neq \text{LSB}(w^\beta)$, we see:

$$x = w \Leftrightarrow \text{LSB}(x^\beta) = \text{LSB}(w^\beta),$$

so we obtain a correct answer with probability exceeding $\frac{1}{2} + \frac{1}{k^{c_1}}$, infinitely often.

Although at first glance the techniques of [1] might seem adaptable to show that we could actually factor n, their argument does not apply directly to the case at hand (with $\text{LSB}(z^\beta)$ rather than $\text{LSB}(z)$), and a fix is not obvious. □

3 Unpredictable Global Bits in Distributed Systems

We turn to the problem of generating global coin flips in a distributed system. This section defines the problem and gives a primitive solution; our solution is found in §4.

Consider a network of N processors ("players"), connected by private channels but lacking a broadcast channel. For simplicity, we assume that a trusted host is available to initialize the network. Both of these assumptions can be weakened by using encryption and initial secret computation protocols, but we leave this to another level of analysis.

At the end of each of B *phases*, each processor should receive a random bit – and all processors should receive the same bit. (We omit the goal of generating more than one bit per phase, as it is a simple modification to our analysis.)

If a trusted host (incorruptible processor) were available, the direct solution would be for it to flip a coin during each phase and simply send the result to each processor. A slightly less direct solution might be for the trusted host to generate a pseudorandom sequence from an initial seed x, and then reveal these bits to the network, one by one. Or it might just send out a string during each phase, from which the current bit can be calculated. The important property is that any attempt to predict the bits will fail.

Definition 4 *An* **ideal URN protocol** *for URN G is a protocol that operates in phases. On input N (network size) and k a trusted host TH selects a k-bit seed at random, and evaluates a poly-strong URN, G. At the end of phases 0 through $B(k)$, TH sends $\hat{G}(k, i, x)$ to all players, who compute their local output, $b(\hat{G}(k, i, x))$.*

Trusted hosts are at best available only for initialization, if at all. Thus we consider networks in which no trusted host is available. If a protocol can be reduced to an ideal URN protocol, via security-preserving reductions such as "relative resilience" or related notions [2, 12], we shall say it is a *global URN protocol.* We quickly outline the formalization.

A protocol α is *as resilient as* a protocol β if there exists an interface $\mathcal{I}$ such that for any adversary $\mathcal{A}$ attacking α, there is an attack by $\mathcal{I}(\mathcal{A})$ on β yielding "equivalent" results. In particular, the vector of random variables describing $\mathcal{A}$'s view and the outputs of nonfaulty players should be indistinguishable (in some sense) in the two scenarios. We refer the reader to [2, 12] for details.

Definition 5 *A* **global bit protocol** *or* **global URN protocol** *is a protocol that is computationally as resilient as an ideal URN protocol for some URN G.*

Definition 6 *A global URN protocol is* **non-interactive** *if each phase (apart from initialization) requires each processor to send at most one message to each other processor (order-independent) per phase.*

Definition 7 *A player is said to* **disseminate** *message m in round ρ if it sends m to all other players in round ρ.*

3.1 A Simple Solution For Constant t

A very simple solution exists when at most a constant number of faults can occur. Interestingly, this solution permits random access to the bits (*ie.* they can be revealed in any order), but nevertheless violates the impossibility result of [8], apparently because the bits are not represented as "shares" in the particular form considered in that work.

For clarity, consider $N = 3$ and $t = 1$, and consider only omission-failures (*ie.* no player sends bad messages, but messages can be dropped). Let G be any pseudorandom bit generator (whether in the sense of the CSPRG's of [6] or an URN as defined above).

To initialize the system, a trusted host generates seeds s_1, s_2, s_3 randomly and gives (s_1, s_2) to player 1, (s_2, s_3) to player 2, and (s_3, s_1) to player 3.

Bit b_j is defined as $G(k, s_1, j) \oplus G(k, s_2, j) \oplus G(k, s_3, j)$.

To generate bit b_j, player 1 disseminates $(G(k, s_1, j), G(k, s_2, j))$, player 2 disseminates $(G(k, s_2, j), G(k, s_3, j))$, and player 3 disseminates $(G(k, s_3, j), G(k, s_1, j))$. Because $t = 1$, even if one pair is omitted, all three values $G(k, s_1, j), G(k, s_2, j), G(k, s_3, j)$ are present. Each player calculates b_j from the appropriate exclusive-or.

Clearly, any $t = 1$ or fewer players cannot predict b_j. It is not hard to see that this technique generalizes for any constant minority t: each $(N - t)$-subset σ is given a seed s_σ from which each member can calculate $G(k, s_\sigma, j)$. Because $N > 2t$, for each σ there is always a nonfaulty player who disseminates $G(k, s_\sigma, j)$ at the appropriate time. On the other hand, for any t-subset τ there is always a σ that excludes τ, so that curious players cannot predict the bit until the appropriate time.

Since t is constant, this protocol uses polynomial time and messages. It uses only dissemination at each round and is thus "non-interactive."

4 Main Result: Fast, Non-interactive Global Bit Generation

Our distributed bit generator is based on the URN $G_{\alpha,\beta}$ described in §2.1, with α and β defined by $2^\alpha \beta = 4(N!)^2$. Based on a secret seed x, the N-player network computes and reveals the following numbers, phase by phase:

$$x^{2^{(B-1)\alpha} \cdot 2(N!)^2}, x^{2^{(B-2)\alpha} \cdot 2(N!)^2}, \ldots, x^{2^{0 \cdot \alpha} \cdot 2(N!)^2},$$

The parities of these numbers are the desired, unpredictable bits. In fact, the parities of these numbers are the bits output by the URN $G_{\alpha,\beta}$. (The strings output by the URN are predecessors (roots) of these values, but the bit extraction function ultimately takes the LSB's of the values shown here.)

In an ideal protocol, a trusted host could broadcast these numbers (or equivalently, the strings given by $G_{\alpha,\beta}$) phase by phase. Without a trusted host, we focus on revealing and agreeing upon each number using simple, non-interactive dissemination (*ie.* in one round without broadcast channels).

For simplicity, let us assume the initialization is taken care of either by a trusted party or an initial multiparty protocol requiring broadcast. Thereafter, neither trust nor broadcast is needed.

```
Initialization($N, t, k, B$)
// $N$: number of players; $t < N/2$: fault tolerance bound;
// $k = \Omega(N \log N)$: security parameter; $B$: number of bits.
    Let $Q$,$M$ be such that $2^Q M = N!$ and $M$ is odd.
    $W \leftarrow 2Q + 2$              // $\alpha = W$, $\beta = M^2$
    $n \leftarrow \textsc{Random}(\textsc{Blum}_k)$
    $g \leftarrow \textsc{Random}(\mathbf{Z}_n^*)$        // see footnote[a]
    $(a, a_1, a_2, \ldots, a_t) \leftarrow \textsc{Random}(\mathbf{Z}_{\phi(n)}^*)^{t+1}$
    Let $f(u) = a + a_1 u + a_2 u^2 + \cdots a_t u^t$
    for $i = 1..N$ do
        $y_i \leftarrow g^{f(i)N!}$          // represents $X = x^{N!} = g^{f(0)N!} = (g^a)^{N!}$
        $z_i \leftarrow y_i^{2^{BW}}$
    for $i = 1..N$ do
        give $(n, y_i, (z_1, z_2, \ldots, z_N))$ to player $i$.
```

[a]The values n and g should be chosen so that the powers of g span a polynomial fraction of $\mathbf{Z}_n^*$, but we omit such considerations for the sake of presentation.

After initialization, each party holds a "share" y_i of a secret value $X = g^{N!a} = x^{N!}$, similar to Feldman's VSS scheme [9]. Any other piece y_j can be "checked" by raising it to an appropriate power and comparing the result to z_j, thus avoiding the need to share the shares for verification purposes.

More to the point, because this scheme is homomorphic, each player i can generate a share of X raised to any desired power, simply by raising y_i to that power. Thus, without interaction, each player can generate shares of $X^{2^{(B-1)W}}, \ldots, x^{2^0}$, simply by computing $y_i^{2^{(B-1)W}}, \ldots, y_i^{2^0}$.

At each phase, each player disseminates its share of the current power of X to be revealed. Unfortunately, it is not clear how to interpolate X (or a power of it) directly from the pieces, since we do not know how to take arbitrary roots. Instead, $X^{2(N!)}$ (and its various powers) are computed. The $N!$ arises from making sure that a unique interpolation can be done over the integers without taking roots (*ie.* division in the exponents). The factor of 2 arises from squaring the revealed pieces to prevent malicious players from disseminating $-y$ instead of y (since the pieces are verified by repeated squaring, an adversary cannot successfully reveal a valid piece other than $\pm y$).

For a (correct) subset $\gamma \subseteq \{1, 2, \ldots, N\}$, define the following functions over $\mathbf{Z}$:

$$L_{i,\gamma}(u) = N! \prod_{j \in \gamma - \{i\}} \frac{u - j}{i - j}.$$

(A straightforward argument shows that each coefficient in $L_{i,\gamma}$ is not only integral but divisible by $i\binom{n}{i}$.) If $f(u)$ is a polynomial of degree t, then as long as γ contains $t+1$ or more elements, we can "interpolate" a unique value:[2]

$$N!f(0) = \sum_{i \in \gamma} f(i) L_{i,\gamma}(0).$$

```
Protocol URNPROTO - code for player $i$
Player($i, N, k, B$)
// $i$: id; $N$: number of players; $k$: security parameter; $B$: number of bits.
    Receive $(n, y_i, \langle z_1, z_2, \ldots, z_N\rangle)$ at startup.
    $\langle v_1, v_2, \ldots, v_N\rangle \leftarrow \langle z_1, z_2, \ldots, z_N\rangle$
    $\gamma = \{1, 2, \ldots, N\}$                    // Correct players.
    for phase $j = 1..B$ do
        { Generate bit $b_j$ }
        Disseminate $y_i^{2^{(B-j)W}}$
        Receive $(w_1, w_2, \ldots, w_N)$ // Apparent pieces
        // Verify pieces against known, higher squares
        $\gamma \leftarrow \{l \in \gamma \mid w_l^{2^W} \equiv v_l\}$
        $(v_1, v_2, \ldots, v_N) \leftarrow (w_1, w_2, \ldots, w_N)$
        // Interpolate $j^{th}$ value $X^{2^{(B-j)W} 2N!}$,
        // using $(w_l)^2$ values to avoid malicious negation.
        $x(j) \leftarrow \prod_{l \in \gamma} w_l^{2L_{l,\gamma}(0)}$
        $b_j \leftarrow \text{LSB}(x(j))$
```

Because an adversary cannot substitute false pieces, the interpolation of $x(j) = X^{2^{(B-j)W} 2N!}$ gives the LSB produced by the URN:

$$\begin{aligned} x(j) &= \prod_{l \in \gamma} w_l^{2L_{l,\gamma}(0)} = g^{\sum_{l\in\gamma} 2L_{l,\gamma}(0)(N!f(l))2^{(B-j)W}} = \left(X^{2(N!)}\right)^{2^{(B-j)W}} \\ &= x^{2^{(B-j)W} 2(N!)^2} = x^{2^{(B-j)W+W-1} M^2}. \end{aligned}$$

4.0.1 An Optimization

Because the bits are output in reverse order, the brute-force computation of the powers of each y_i would require $\Theta(B^2)$ repeated-squarings, or B per bit. Alternatively, storing the intermediate results would use $\Theta(B)$ space – a factor whose disadvantages themselves motivate using pseudorandom number generators rather than "one-time pads."

In the full paper, we present a simple technique we call "signposting," which uses a total of $\Theta(B \log B)$ repeated-squarings, or $\log B$ repeated-squarings per bit, but $\Theta(\log B)$ space overall. This algorithmic optimization is merely a faster local computation; it does not change the protocol in any way.

4.0.2 A Second Solution

At the cost of some extra exponentiation, we can base our results on the weaker assumption that factoring is intractable. Letting $x = g^a$, observe that $G_{\alpha,\beta}$ outputs numbers of the form

$$x(j) = x^{2^{(B-j)W+W-1} M^2}.$$

[2]The value we obtain is unique regardless of γ; but it is $N!f(0)$, *not* $f(0)$.

Because of the odd factor M^2 in the exponent, we have thus far only been able to base unpredictability on the QRA. If, instead, we use numbers of the form

$$v(j) = x^{2^{(B-j)W+W-1}},$$

namely if we omit the odd factors in the exponent, then there is a straightforward modification of the arguments of §2.1 and [1] to show that predicting the LSB's is as hard as factoring. Let us call this second generator $V_{\alpha,\beta}$.

The problem is that the polynomial interpolation used in the secret sharing introduces an $N!$ factor in the exponent, apparently making the odd power unavoidable. But let us say that the number $u(j-1) = x^{2^{(B-(j-1))W}}$ is available – this next higher "power of two" already played a role in the bits revealed earlier. Then $v(j)$ can be calculated from $x(j)$ and $u(j-1)$:

$$v(j) = x(j)/u(j-1)^{\frac{M^2-1}{2}}$$

as is demonstrated by the following:

$$x(j)/u(j-1)^{\frac{M^2-1}{2}} = x^{2^{(B-j)W+W-1}M^2-(2^{(B-(j-1))W})\frac{M^2-1}{2}} = x^{2^{(B-j)W+W-1}}.$$

Thus, at additional cost, we can in fact calculate a "pure" power-of-two exponent of the original seed. The cost is one division and an exponentiation to the power $\frac{M^2-1}{2}$, where M is the odd part of $N!$. If we make $u(0) = x^{2^{BW}}$ available at the start, subsequent $u(j)$'s can be derived using the same trick as shown above.

4.1 Proof of Security

Theorem 3 *Protocol* URNPROTO *is a non-interactive global bit protocol, unless the QRA fails.*

Proof. Non-interaction holds by definition. We show a security reduction from URNPROTO to an ideal URN protocol (call it THURN) in which the trusted host broadcasts the sequence generated by the URN $G_{\alpha,\beta}$ of §2.1. For clarity, we consider only the case of a static adversary (the set of faulty players is chosen in advance). Given an adversary $\mathcal{A}$, we must show how an interface $\mathcal{I}$ maps $\mathcal{A}$'s attacks on URNPROTO to attacks on THURN. The main job of $\mathcal{I}$ is to construct initial $(y_i, (z_1, z_2, \ldots, z_N))$ vectors for corrupted players and to simulate the shares $y_i^{2^{(B-j)W}}$ disseminated in each phase j by nonfaulty processors. We show how $\mathcal{I}$ does this accurately, conditioned on $\mathcal{A}$ being unable to take square roots (a condition violated with negligible probability).

In short, $\mathcal{I}$ selects $\hat{a}, \hat{a}_1, \ldots, \hat{a}_t$ at random, sets $\hat{f}(u) = \hat{a} + \hat{a}_1 u + \cdots + \hat{a}_t u^t$, and for each faulty player i computes the value

$$h_i = g^{\hat{f}(i)}.$$

Note that $\mathcal{I}$ cannot compute "real" shares of a polynomial with free term a because it will not receive g^a in the ideal protocol until it's too late. Now, when player i becomes corrupted, $\mathcal{I}$ pretends i's piece is $y_i = h_i^{N!}$. This will cause no problem, as long as: (1) $\mathcal{A}$ never substitutes an undetected but incorrect piece (± 1 factors are acceptable); (2) $\mathcal{I}$ can accurately generate pieces disseminated by nonfaulty processors. Event (1) occurs with negligible probability (else factoring is easy), so we focus on (2).

At round j, then, $\mathcal{I}$ must generate fake but convincing values $y_i^{2^{(B-j)W}}$ from nonfaulty players. The $N!$ interpolation factor would normally prevent $\mathcal{I}$ from directly interpolating consistent pieces, but the initial exponentiation (representing $X = x^{N!}$ rather than x) permits $\mathcal{I}$ to get away with introducing an $N!$ factor. Interpolating from $h_1^{2^{(B-j)W}}, \ldots, h_t^{2^{(B-j)W}}$ and $x^{2^{(B-j)W}}$, $\mathcal{I}$ computes $y_i = g^{N! f(i) 2^{(B-j)W}}$ where the polynomial $f(u)$ agrees with $\hat{f}(u)$ at t

places but satisfying $f(0) = a$. This provides fake pieces from correct players at any phase (including the initial verification values, z_i).

The result is a distribution identical to that obtained when $\mathcal{A}$ attacks URNPROTO, conditioned on event (1) not occurring. When event (1) is included, the distributions are statistically indistinguishable. Intuitively, we have shown that an attack by $\mathcal{A}$ on URNPROTO achieves nothing that an attacker against the ideal trusted host protocol can't achieve – and in particular, predicting the bits is impossible. □

5 Discussion

We have presented two unpredictable bit generators requiring neither broadcast nor interaction other than simple dissemination (after an interactive initialization stage). The local computation is reasonably fast – repeated squaring, like the BBS generator, along with "interpolation" that requires some exponentiation – yet there is no need for "shares of shares" or other complicated constructs. A share is verified simply by squaring it.

Our bit generators differ from the secret bit generators discussed in [8] in that they do not provide "random access" (the constant-t generator of §3.1 is a notable exception), and they focus on unpredictability rather than the subtly different notion of pseudorandomness.

Although the unpredictability of the simpler of our generators is based on the QRA, we conjecture that it is in fact unpredictable unless factoring is easy. The methods of [1] do not seem to apply immediately, and further work is required.

We also conjecture that the number of squarings can be reduced by 50%; our protocol includes an apparently superfluous $N!$ factor to facilitate the *proof* of security.

Recently, a fast, non-interactive, global URN based on elliptic curves over rings has been developed and investigated in [3]. This solution shares the property of ours that extra "verification" procedures (such as shares of shares) are not needed. As shown in [8] for the case of doubly shared bits, we conjecture that our methods will apply to any homomorphic scheme.

References

[1] W. Alexi, B. Chor, O. Goldreich, C.P. Schnorr. "RSA and Rabin Functions: Certain Parts are as Hard as the Whole." *SIAM J. Computing,* **17**:2 (1988), 194–209.

[2] D. Beaver. "Foundations of Secure Interactive Computing." *Proc. of Crypto 1991*, 377–391.

[3] D. Beaver, H. Shan. In preparation.

[4] M. Ben-Or. "Another Advantage of Free Choice." *Proc. of* 2^{nd} *PODC,* 1983.

[5] Blakley, "Security Proofs for Information Protection Systems." *Proceedings of the the 1980 Symposium on Security and Privacy,* IEEE Computer Society Press, NY (1981), 79–88.

[6] M. Blum, S. Micali. "How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits." *SIAM J. Comput.* **13** (1984), 850–864.

[7] L. Blum, M. Blum, M. Shub. "A Simple Unpredictable Pseudo-Random Number Generator." *SIAM J. Computing,* **15**:2 (1986), 364–383.

[8] M. Cerecedo, T. Matsumoto, H. Imai. "Non-Interactive Generation of Shared Pseudorandom Sequences." To appear, Auscrypt 92.

[9] P. Feldman. "A Practical Scheme for Non-Interactive Verifiable Secret Sharing." *Proc. of the* 28th *FOCS,* IEEE, 1987, 427–437.

[10] O. Goldreich, S. Goldwasser, S. Micali. "How to Construct Random Functions." *JACM* **33**:4 (1986), 792–807.

[11] S. Goldwasser, S. Micali. "Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information.

[12] S. Micali, P. Rogaway. "Secure Computation." *Proc. of Crypto 1991*, 392–404.

[13] M. Pease, R. Shostak, L. Lamport. "Reaching Agreement in the Presence of Faults." *JACM* **27**:2 (1980), 228–234.

[14] T. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing." Proceedings of CRYPTO 91, 129–140.

[15] M.O. Rabin. "Digitalized Signatures and Public-Key Functions as Intractable as Factorization." Technical Report LCS/TR-212, MIT, January, 1979.

[16] M.O. Rabin. "Randomized Byzantine Generals." *Proc. of the* 24th *FOCS,* IEEE, 1983, 403–409.

[17] A. Shamir. "How to Share a Secret." *Communications of the ACM,* **22** (1979), 612–613.