Note on Secret Sharing and SMC

Patrick Ah-Fat

Contents

1 Motivations **1**
- 1.1 Introduction . . . 1
- 1.2 Adversaries . . . 2

2 Secret Sharing **3**
- 2.1 Monotone Access Structures . . . 3
- 2.2 Threshold Structures . . . 4
- 2.3 Lagrange Interpolation . . . 4
- 2.4 Shamir Secret Sharing Scheme . . . 6

3 SMC with Passive Security **7**

References **10**

1 Motivations

1.1 Introduction

Secret Sharing (SS) is a domain of cryptography which enables a dealer to share a secret amongst a set $\mathcal{P}$ of parties such that only designated subsets of $\mathcal{P}$ can recover the secret, while any other subset of parties cannot learn anything about it [7, 2]. Different methods have been proposed to this end, including Ito, Saito and Nishizeki's scheme and the replicated secret sharing scheme [5]. In this document, we will study a so called "t out of n" threshold scheme that allows a shared secret to be recovered by any t parties while any set of less than t parties cannot infer anything.

On the other hand, Secure Multi-Party Computation (SMC) enables several parties to compute a public function of their own private inputs while maintaining the confidentiality of the inputs and without resorting to any trusted third party. The protocols that implement SMC fall into two main categories. The first one gathers the methods based on Yao's garbled circuits [9] and oblivious transfer [6]. The second one, which we present in this document, contains the protocols based on secret sharing. The main interest is that those schemes are information-theoretically secure whereas Yao's protocols build on oblivious transfer and are thus only computationally secure. Moreover, Yao's garbled circuits were specifically designed and are thus particularly suited for two-party computations of boolean functions. On the contrary, Secret Sharing naturally produces a more general class of SMC protocols that handle multi-party computations of arithmetic functions. Finally, Secret Sharing is a rich topic in itself which makes use of powerful algebraic properties that are closely related to other fields in Computing such as Error-Correcting Codes.

We formalise the notion of adversary in Section 1.2. We introduce Secret Sharing in Section 2 and we focus on Shamir's scheme in Section 2.4. Based on this scheme, we construct an SMC protocol with passive security in Section 3.

Notations: Let $\mathbb{F}$ be a field. Let t be in $\mathbb{N}$. We denote by $\mathbb{F}_t[X]$ the set of polynomials over $\mathbb{F}$ of degree at most t. For any polynomial f in $\mathbb{F}_t[X]$, the degree of f is denoted by $\deg(f)$. For any positive integer n, we denote by $\mathbb{Z}_n$ the set of equivalence classes modulo n. The cardinality of a finite set S is denoted by $|S|$ and $\mathscr{P}(S)$ denotes its power set. For any two integers a and b, $[\![a, b]\!]$ denotes the set of consecutive integers ranged between a and b, namely $\{a, a+1, \cdots, b\}$.

1.2 Adversaries

In both Secret Sharing and Secure Multi-party Computation, the main objective is to perform a task — storage of a secret or computation of a function — while protecting the privacy of the secrets against a number of potentially malicious parties. The notion of adversary is thus inherent to the concepts of SS and SMC. For this reason, we need to formalise the power of the adversaries and to make their capabilities explicit. Based on those definitions, we will then be able to build bespoke protocols that are resilient against a number of dishonest participants who may jeopardise the protocols proceedings.

Definition 1. *We can distinguish two main types of adversaries.*

- *A* passive adversary *or honest-but-curious adversary is a party who abides by the protocol but who will share with other adversaries all the information that she receives during the protocol so as to infer as much information as possible on the secrets.*

- *An* active adversary *or malicious adversary can not only share the information that she receives with other adversaries, but she can also try to deviate from the protocol rules and to cooperate with other adversaries in order to gain more information about the secrets, or to corrupt the protocol outputs.*

We will also refer to adversaries as attackers. We generally assume that all the present adversaries have the same power. A scheme that is robust against passive adversaries will be denoted as passively secure, while an actively secure scheme will denote a scheme that is resilient against active adversaries. Naturally, passively secure schemes will be conceptually and computationally less heavy, but actively secure schemes will have much stronger guarantees. For these reasons, this document will focus on some methods for achieving SS and SMC schemes in the presence of passive adversaries only.

Example 1. *Let us consider a simple secret sharing scheme involving three parties Ann, Bob and Claire where Ann wants to secretly share her secret integer a. She decides to split her secret into two integer shares a_B and a_C such that $a = a_B + a_C$ and sends a_B to Bob and a_C to Claire. Recovering the value of a would require Bob and Claire to send their shares back to Ann who would then compute their sum. Then:*

- *If only Bob is a passive attacker, then his influence is limited: he cannot infer any information about the secret a from his own share a_B.*
- *If both Bob and Claire are passive attackers, then they can together recover the secret a since they together hold both a_B and a_C.*
- *If Bob is an active attacker, he can corrupt the secret recovery phase. Indeed, instead of sending his share a_B back to Ann, he can send a random integer, which would lead Ann to recover an erroneous secret.*

2 Secret Sharing

2.1 Monotone Access Structures

A secret sharing scheme allows a dealer to share her secret amongst several players. When designing a secret sharing scheme, we need to specify the subsets of parties that will be able to recover the secret. A set of parties that can together recover the secret is called *qualified*. The *access structure* of a secret sharing scheme is the set of all qualified sets of parties.

We can sensibly assume that access structures are monotone, i.e. that any superset of a qualified set is qualified. In other words, if some parties are able to recover the secret, then they will also be able to recover the secret with additional parties.

Example 2. *Let us consider* 4 *parties Ann, Bob, Claire and Dan. Let us imagine a secret sharing scheme that would allow any sets of parties containing Ann and Bob to recover the secret. Additionally, any sets of parties containing Ann and Claire should be able to recover the secret. Any other set of parties should not be able to infer any information about the secret.*

By denoting each party by their initial, the access structure $\mathcal{A}$ of such a scheme would be:

$$\mathcal{A} = \{\{A, B\}, \{A, B, C\}, \{A, B, D\}, \{A, B, C, D\}, \{A, C\}, \{A, C, D\}\}$$

We can more conveniently represent $\mathcal{A}$ by the set of its minimal elements $m(\mathcal{A})$ with respect to subset inclusion:

$$m(\mathcal{A}) = \{\{A, B\}, \{A, C\}\}$$

Definition 2. *Let $\mathcal{P}$ be a set of parties. Let $\Gamma \subseteq \mathscr{P}(\mathcal{P})$ be a non-empty collection of subsets of $\mathcal{P}$. Then Γ is a monotone access structure if and only if:*

$$\forall A \subseteq B \subseteq \mathcal{P} \colon (A \in \Gamma) \implies (B \in \Gamma)$$

We notice that the definition implies that for any monotone access structure Γ on $\mathcal{P}$, we have $\mathcal{P} \in \Gamma$, i.e. that all parties together can recover the secret.

2.2 Threshold Structures

An important class of monotone access structures is that of *threshold structures*.

Definition 3. *Let $1 \leq t \leq n$ be two positive integers. Let $\mathcal{P} = \{P_1, \cdots, P_n\}$ be a set of n parties. The t-out-of-n monotone access structure Γ is defined by its minimal elements as follows:*

$$m(\Gamma) = \{S \subseteq P \mid |S| = t\}$$

Informally, for any two integers $1 \leq t \leq n$, the t-out-of-n monotone access structure is the set of all subsets of at least t parties.

General secret sharing schemes such as Ito, Saito and Nishizeki's scheme or the replicated secret sharing scheme [5] can accommodate any type of monotone access structures. However, those schemes are inefficient in terms of the number of shares needed to distribute a secret, and they do not naturally adapt to the presence of dishonest parties. In the following sections, we will study Shamir secret sharing scheme which is specifically designed for threshold structures, and which overcome those problems. The next section presents some recalls on Lagrange interpolation, which is required for Shamir SS scheme.

2.3 Lagrange Interpolation

Let us recall that, for any t in $\mathbb{N}$, Lagrange interpolation enables us to recover any polynomial of degree at most t given at least $t + 1$ of its points. In particular, given at least $t + 1$ points of a polynomial f of degree at most t, we can reconstruct $f(0)$, as we formalise in the next Claim.

Claim 1. *Let $\mathbb{F}$ be a field (not necessarily finite). Let t be in $\mathbb{N}$. Let f be a polynomial of degree at most t in $\mathbb{F}_t[X]$.*

Then, for all subset S of $\mathbb{F}$ with more than t elements, there exists a vector $r = (r_k)_{k \in S}$ in $\mathbb{F}^{|S|}$, called recombination vector, satisfying:

$$f(0) = \sum_{k \in S} r_k f(k)$$

Proof. For all element i in S, we recall that the Lagrange basis polynomial δ_i is defined as:

$$\delta_i(X) = \prod_{j \in S, j \neq i} \frac{X - j}{i - j} \tag{1}$$

We notice that for a given i in S, the polynomial δ_i verifies $\delta_i(i) = 1$, and for all j in S different from i, we have $\delta_i(j) = 0$. The Lagrange polynomial g is then defined as:

$$g(X) = \sum_{i \in S} f(i)\delta_i(X) \tag{2}$$

Then, Lagrange interpolation claims that $f = g$. Indeed, the polynomial $f - g$ is of degree at most t and clearly evaluates to 0 on the $t+1$ or more points of S and thus is the zero polynomial by the Fundamental Theorem of Algebra.

Let us now define the recombination vector $r = (r_k)_{k \in S}$ in $\mathbb{F}^{|S|}$ where for all k in S, we define $r_k = \delta_k(0)$. Then, Equation (2) ensures that:

$$f(0) = \sum_{k \in S} r_k f(k) \tag{3}$$

which not only concludes the proof, but also provides us with a method of reconstructing $f(0)$ given more than t points of f. □

Example 3. *Let us consider the finite field $\mathbb{F} = \mathbb{Z}_{11}$. Let us consider a polynomial $f \in \mathbb{F}_2$ over $\mathbb{F}$ and of degree t at most 2. We assume that we have the following values of the polynomial: $f(2) = 7, f(4) = 3, f(5) = 4, f(6) = 7, f(10) = 6$. We have 5 points of this polynomial of degree at most 2. As $5 > 2$, we can reconstruct f, and in particular $f(0)$, via Lagrange interpolation.*

We also know that 3 points are enough to recover f. So let us choose $S = \{2, 4, 5\}$, a set of values whose image by f is known. We have $|S| > t$, so let us now reconstruct the value $f(0)$. Let us compute the recombination vector for this set S. For all i in S, Equation (1) gives us:

$$\delta_i(X) = \prod_{j \in S, j \neq i} \frac{X - j}{i - j}$$

and thus:

$$\delta_i(0) = \prod_{j \in S, j \neq i} \frac{-j}{i - j}$$

So:

$$\delta_2(0) = \frac{-4}{2-4} \cdot \frac{-5}{2-5} = \frac{-4}{-2} \cdot \frac{-5}{-3} = 10 \cdot 3^{-1} = 10 \cdot 4 = 7$$

Similarly:

$$\delta_4(0) = \frac{-2}{4-2} \cdot \frac{-5}{4-5} = -1 \cdot 5 = 6$$

and:

$$\delta_5(0) = \frac{-2}{5-2} \cdot \frac{-4}{5-4} = 8 \cdot 3^{-1} = 8 \cdot 4 = 10$$

Equation (3) thus yields:

$$f(0) = \sum_{k \in S} r_k f(k) = 7 \cdot 7 + 6 \cdot 3 + 10 \cdot 4 = 5 + 7 + 7 = 8$$

One should note that:

1. *Although we could reconstruct the polynomial $f(X)$, we do not need it to reconstruct $f(0)$.*
2. *We only need $t+1$ points to reconstruct the secret. Considering larger sets S would yield the same result with more complex calculations.*

2.4 Shamir Secret Sharing Scheme

We are now able to present Shamir secret sharing scheme [7]. Let n be a positive integer and let us consider a set of n parties $\mathcal{P} = \{P_1, \cdots, P_n\}$. Let t be in $\mathbb{N}$ such that $t < n$. We will allow the presence of t passive adversaries.

Let p be a prime number. We now consider the finite[1] field $\mathbb{F} = \mathbb{Z}_p$. Let us consider the set $S = [\![1, n]\!]$. Let d be in $[\![1, n]\!]$ and let us consider a secret value a in $\mathbb{Z}_p$ held by a certain party P_d, the dealer. In order to share her secret, party P_d execute the following steps.

Shamir Secret Sharing Scheme: Distribution [7]

1. Party P_d chooses a random polynomial f_a of degree at most t (where $t < n$) such that $f_a(0) = a$. In other words, she chooses t random coefficients $(a_k)_{k \in [\![1,t]\!]}$ in $(\mathbb{Z}_p)^t$, she sets the first coefficient to her secret value $a_0 = a$ and builds the resulting polynomial $f_a(X) = \sum_{k=0}^{t} a_k X^k$.
2. Then, for all k in $[\![1, n]\!]$, party P_d sends $f_a(k)$ to party P_k, where $f_a(k)$ will be considered as one share of secret a.

Once those steps have been performed, we say that P_d has distributed $[a, f_a]_t$. We notice that Lagrange interpolation not only ensures that any set of more than t parties can recover the the polynomial f_a and thus the secret a, but it also guarantees that any set of t or less than t parties cannot infer anything about the secret value a.

Shamir Secret Sharing Scheme: Reconstruction [7]

Let $S \subseteq \mathcal{P}$ be a subset of more than t parties wishing to reconstruct the secret. By virtue of Claim 1 and as $|S| > t$, let us define the recombination vector $r = (r_k)_{k \in S}$ in $(\mathbb{Z}_p)^n$ which ensures that for all polynomial f of degree at most t, we have $f(0) = \sum_{k \in S} r_k f(k)$. In order to reconstruct the

[1] Shamir secret sharing scheme is based on Lagrange interpolation and thus works for any choice of field. For example, the theory would also work on $\mathbb{R}$. However, when implementing the scheme in practice, we need to store the exact value of the shares on a machine, which is why we require the field to be *finite*. We choose to work in $\mathbb{Z}_p$ here for simplicity purposes.

shared secret, the parties will simply have to open the value of the share that they received, and compute $a = f_a(0) = \sum_{k \in S} r_k f_a(k)$.

Example 4. *Let us consider* 6 *parties* $P_1, \cdots, P_6$. *Let us assume that one of the parties, say party* P_3, *has a secret* $s = 8$ *that she wants to share amongst the parties, such that any set of more than* 2 *parties should be able to recover the secret, whereas any set of less than* 3 *parties should not be able to infer anything about the secret. Let us place ourselves in the finite field* $\mathbb{Z}_{11}$ *again.*

She can use Shamir secret sharing scheme to this end: she generates a random polynomial f *of degree at most* 2 *such that* $f(0) = 8$. *In other words, she chooses two coefficients* a_1 *and* a_2 *at random, say* $a_1 = 3$ *and* $a_2 = 1$ *and she sets* $a_0 = 8$ *to her secret value. She then builds the polynomial* f *as:*

$$f = a_0 + a_1 X + a_2 X^2 = 8 + 3X + X^2$$

Now, for all party P_k, *she sends the share* $f(k)$ *to party* P_k, *i.e. she sends* $f(1) = 1$ *to* P_1, $f(2) = 7$ *to* P_2, $f(3) = 4$ *to* P_3 *(i.e. to herself),* $f(4) = 3$ *to* P_4, $f(5) = 4$ *to* P_5 *and* $f(6) = 7$ *to* P_6. *At this stage, we say that the parties secretly share* $[8, f]_2$, *which means that they share the secret value* 2 *with a polynomial* f *that is of degree at most* 2.

Now any set of more than 2 *parties would be able to recover the secret. For example,* P_2, P_4 *and* P_5 *can recover the secret, in the way presented in Example 3.*

3 SMC with Passive Security

In this section, we will show that we are able to perform Secure Multi-party Computation (SMC) by manipulating secretly shared values [1]. Here again, we will allow the presence of t passive adversaries. However, we will now assume that $t < n/2$. This assumption will have an importance in processing multiplication gates.

First, let us now see how the parties can manipulate their shares to perform operations on secrets. Let a and b be two elements of $\mathbb{Z}_p$ held secretly by two given parties. Let f_a and f_b be two polynomials of degree at most t such that $f_a(0) = a$ and $f_b(0) = b$. We further assume that those values have been secretly shared and that the parties hold $[a, f_a]_t$ and $[b, f_b]_t$. In other words, this means that for all k in $[\![1, n]\!]$, party P_k has received both values of $f_a(k)$ and $f_b(k)$. Finally let γ be a constant and public value in $\mathbb{Z}_p$. Next, we show that the parties are able to secretly share the values of $a + b$, γa and $a * b$ via polynomials of degree at most t, without revealing any information on a and b.

- We notice that the parties can easily share the sum of a and b via a polynomial of degree at most t. Indeed, for all k in $[\![1, n]\!]$, let party P_k locally compute the sum of her shares $f_a(k) + f_b(k)$, which is incidentally equal to $(f_a + f_b)(k)$. Then, the parties now share the value $a + b$ via the polynomial $f_a + f_b$. As the degree of $f_a + f_b$ is at most t, we say that the parties share $[a+b, f_a+f_b]_t$, and by abuse of notation, we say that the parties have computed $[a, f_a]_t + [b, f_b]_t = [a+b, f_a+f_b]_t$.

Now, any set of more than t parties can recover the value of $a+b$ while any set of t or less than t parties cannot infer anything about $a+b$. The interesting thing is that when reconstructing the value of $a+b$, the parties will not learn anything on the values of a and b.

- Similarly, the parties can easily compute multiplications by a constant. For all k in $[\![1, n]\!]$, let party P_k locally compute $\gamma * f_a(k)$, which is obviously equal to $(\gamma * f_a)(k)$. The parties now share the secret value $\gamma * a$ via the polynomial γf_a of degree at most t, and we say that the parties have computed $\gamma * [a, f_a]_t = [\gamma a, \gamma f_a]_t$.
- The multiplication of two shared secrets is slightly more subtle and is worth a protocol on its own.

Protocol Secure Multiplication [3]

1. For all k in $[\![1, n]\!]$, let party P_k locally compute $f_a(k) * f_b(k)$. This enables the parties to share $[a, f_a]_t * [b, f_b]_t = [ab, f_a f_b]_{2t}$, i.e. the parties now share the secret ab via the polynomial $f_a f_b$ of degree at most $2t$. The parties execute the following steps in order to share the product ab via a polynomial of degree at most t.
2. For all k in $[\![1, n]\!]$, party P_k secretly shares her share $(f_a f_b)(k)$ via a random polynomial g_k of degree at most t, i.e. she distributes $[(f_a f_b)(k), g_k]_t$.
3. The parties now invoke the methods for performing addition and multiplication by a constant previously described in order to compute:
$$\sum_{k=1}^{n} r_k[(f_a f_b)(k), g_k]_t = [\sum_{k=1}^{n} r_k (f_a f_b)(k), \sum_{k=1}^{n} r_k g_k]_t$$
where $r = (r_1, \cdots, r_n)$ is the recombination vector introduced earlier which ensures that $\sum_{k=1}^{n} r_k (f_a f_b)(k)$ equals $(f_a f_b)(0)$ since the degree of $f_a f_b$, is at most $2t$ and is thus lower than n (as t is assumed to be lower than $n/2$). As we have $(f_a f_b)(0) = ab$, the players now share the product ab via a polynomial of degree at most t, i.e. they hold $[ab, \sum_{k=1}^{n} r_k g_k]_t$.

We now know that the parties can perform operations on shared secrets while maintaining the degree of the polynomials used to share those secrets lower or equal to t. We can now describe a protocol for securely evaluating n-ary arithmetic functions. Let $x_1, \cdots, x_n$ be n elements of $\mathbb{Z}_p$. For all k in $[\![1, n]\!]$, we assume that x_k is the secret input of party P_k. Let f be an n-ary arithmetic function to be evaluated.

Protocol SMC with passive security [1]

1. Input sharing: For all k in $[\![1, n]\!]$, party P_k secretly shares her input x_k via a random polynomial f_k of degree at most t, i.e. she distributes $[x_k, f_k]_t$.

2. Evaluation: For each elementary operation of f, the parties invoke the methods for computing addition, multiplication by a constant and multiplication so as to secretly share the result of the intermediate computations via polynomials of degree at most t.

3. Output reconstruction: At this stage, the parties secretly share the output value o in $\mathbb{Z}_p$ with a polynomial f_o of degree at most t, i.e. they hold $[o, f_o]_t$. All the parties open their share of o, i.e. for all k in $[\![1, n]\!]$, party P_k opens $f_o(k)$ and the parties reconstruct the output o with the recombination vector r as $o = \sum_{k=1}^{n} r_k f_o(k)$.

Example 5. *Let us illustrate the multiplication protocol. Let us place ourselves in $\mathbb{Z}_{11}$ again. We consider 3 parties P_1, P_2 and P_3. First of all, by virtue of Claim 1, we know that there exists a recombination vector $(r_k)_{1 \leq k \leq 3}$ which ensures that for all polynomial g of degree at most 2, we have $\sum_{k=1}^{3} r_k g(k)$ equals $g(0)$. Let us compute such a vector.*

By setting $S = \{1, 2, 3\}$, Equation (1) yields:

$$r_1 = \frac{-2}{1-2} \cdot \frac{-3}{1-3} = 3$$
$$r_2 = \frac{-1}{2-1} \cdot \frac{-3}{2-3} = 8$$
$$r_3 = \frac{-1}{3-1} \cdot \frac{-2}{3-2} = 1$$

We recall that this recombination vector r only depends on the set S of parties entering the computation and is independent of the function to be computed. It is thus a public and constant value. Finally, for a given polynomial f of degree at most t which evaluates to some value s in 0, we recall that the parties secretly share $[f, s]_t$ means that each party P_k locally holds the share $f(k)$. For readability purposes and in order to make those shares explicit, we will then say that the parties secretly share $[s, f(1), f(2), f(3)]_t$. In other words, this vectorial representation means that P_1 holds $f(1)$, $P(2)$ holds $f(2)$, P_3 holds $f(3)$, and that the secret is s, and is shared with an underlying polynomial of degree at most t. This vectorial representation will also allow us to perform local additions and multiplication by constants intuitively.

We allow the presence of at most $t = 1$ active adversary, and we can check that $t < n/2$. We now assume that those three parties share two values $a = 5$ and $b = 7$ with two respective polynomials $f_a = 5 + 2X$ and $f_b = 7 + 6X$ of degree at most 1. The parties thus secretly share $[5, 7, 9, 0]_1$ and $[7, 2, 8, 3]_1$. We will now apply each step of the multiplication protocol.

1. *The players locally multiply their shares in order to get* $[2, 3, 6, 0]_2$.
2. *Each party will now generate a random polynomial of degree at most* 1 *in order to redistribute the product that she has just computed in the previous step:*
   - P_1 *generates* $g_1 = 3 + X$ *and distributes* $[3, 4, 5, 6]_1$.
   - P_2 *generates* $g_2 = 6$ *— yes she can — and distributes* $[6, 6, 6, 6]_1$.
   - P_3 *generates* $g_3 = 2X$ *and distributes* $[0, 2, 4, 6]_1$.
3. *Each party* P_j *now locally computes* $\sum_{k=1}^{3} r_k g_k(j)$ *so that they share:*

$$3[3, 4, 5, 6]_1 + 8[6, 6, 6, 6]_1 + 1[0, 2, 4, 6]_1 = [2, 7, 1, 6]_1$$

The parties now share the product $ab = 2$ *with a polynomial of degree at most* 1*, which can be shown to equal* $2 + 5X$.

References

[1] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10. ACM, 1988.

[2] George Robert Blakley. Safeguarding cryptographic keys. In *Proc. AFIPS 1979 National Computer Conference*, pages 313–317, 1979.

[3] Ronald Cramer, Ivan Damgård, and Ueli Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *Advances in CryptologyEUROCRYPT 2000*, pages 316–334. Springer, 2000.

[4] Ronald Cramer, Ivan Bjerre Damgard, and Jesper Buus Nielsen. Secure multiparty computation and secret sharing. 2015.

[5] Mitsuru Ito, Akira Saito, and Takao Nishizeki. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 72(9):56–64, 1989.

[6] Michael O Rabin. How to exchange secrets with oblivious transfer. 1981.

[7] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[8] Lloyd R Welch and Elwyn R Berlekamp. Error correction for algebraic block codes, December 30 1986. US Patent 4,633,470.

[9] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.