3 Simplification to Secure Multiparty Computations

We consider the problem of *secure multiparty computation* [Yao82, GMW87, BGW88, CCD88]. There are n players $P_1, \ldots, P_n$. Player P_i holds an input x_i and the players want to compute a function $F(x_1, \ldots, x_n)$ in a *secure* manner, which intuitively means that the adversary cannot disrupt the computation, i.e. the value computed is correct, furthermore the adversary does not learn any information about the inputs of the good players (except for what is revealed by the function value).

MODEL AND DEFINITIONS. We consider a synchronous model with private channels and broadcast (e.g.[RB89, Bea89]). The parties engage in a distributed computation, following a protocol π, in order to evaluate $F(x_1, ..., x_n)$. We assume that there is an adversary $\mathcal{A}$ that corrupts up to t players and coordinates their actions in an arbitrary manner. The adversary we consider is *static* i.e. it decides which players to corrupt at the beginning of the computation. Also our adversary is computationally unbounded. We follow formal definitions of secure multiparty computations that have appeared in several papers [MR91, Bea91, CFGN96, Can95].

In this section we will describe two simplifications to the [BGW88] protocol, and in particular to the multiplication protocol. We first describe an algebraic simplification followed by a simplified zero-knowledge proof for a specific property.

3.1 Algebraic Simplification for Multiplication Protocol

In the following we shall present a simple method for computing the multiplication of two secrets which are distributed among a set of players.

Given two secrets α and β shared by polynomials $f_\alpha(x)$ and $f_\beta(x)$ respectively of degree t, the players would like to compute the product $\alpha\beta$. In their seminal paper Ben-Or et al. [BGW88] note that it isn't sufficient for each player to locally multiply his shares of both secrets, as this generates a polynomial whose constant term is the desired one, i.e. $\alpha\beta$, but it is of degree $2t$ and is not a random polynomial. To overcome this they introduced a degree reduction and randomization protocols. We will show how to achieve both the degree reduction and the randomization in a single step. This building block can be substituted for the multiplication step in the protocol of [BGW88], as it works in the same model of computation. The computation in this section is described under the assumption that all players act properly (as has been said, methods for how to remove this assumption appear in the next section).

Denote by $f_\alpha(i)$ and $f_\beta(i)$ the shares of player P_i on $f_\alpha(x)$ and $f_\beta(x)$ respectively.

The product of $f_\alpha(x)$ and $f_\beta(x)$ is $f_\alpha(x)f_\beta(x) = a_{2t}x^{2t} + ... + a_1x + \alpha\beta \stackrel{\text{def}}{=} f_{\alpha\beta}(x)$. For $1 \leq i \leq 2t+1$, $f_{\alpha\beta}(i) = f_\alpha(i)f_\beta(i)$. Thus we can write

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 2 & \cdots & 2^{2t} \\ \vdots & & & \\ 1 & 2t+1 & \cdots & 2t+1^{2t} \end{bmatrix} \begin{bmatrix} \alpha\beta \\ a_1 \\ \vdots \\ a_{2t} \end{bmatrix} = \begin{bmatrix} f_{\alpha\beta}(1) \\ f_{\alpha\beta}(2) \\ \vdots \\ f_{\alpha\beta}(2t+1) \end{bmatrix}$$

Denote the above matrix by A. This is a $2t+1$ by $2t+1$ Van der Monde matrix, hence non-singular and has an inverse. Let the first row of the inverse matrix, A^{-1}, be $(\lambda_1, ..., \lambda_{2t+1})$, note that these are known constants. Then the previous equation implies that $\alpha\beta = \lambda_1 f_{\alpha\beta}(1) + ... + \lambda_{2t+1} f_{\alpha\beta}(2t+1)$.

Given polynomials $h_1(x), ..., h_{2t+1}(x)$ all of degree t which satisfy that $h_i(0) = f_{\alpha\beta}(i)$ for $1 \leq i \leq 2t+1$, define $H(x) \stackrel{\text{def}}{=} \sum_{i=1}^{2t+1} \lambda_i h_i(x)$. Note that $H(0)$ is exactly $\lambda_1 f_{\alpha\beta}(1) + ... + \lambda_{2t+1} f_{\alpha\beta}(2t+1)$ and hence $\alpha\beta$. Furthermore, $H(j) = \sum_{i=1}^{2t+1} \lambda_i h_i(j)$.

The polynomial $H(x)$, used for the sharing of $\alpha\beta$ is automatically of degree t. It is random because the λ_i are non-zero (easy to check by inspection) and there are $n - t$ polynomials $h_i(x)$ chosen by good players, and hence at random. Thus, the sharing of $\alpha\beta$ by a random polynomial of degree t can be achieved directly following Protocol Simple-Mult in Figure 2.

Simple-Mult

Input of Player P_i: The values $f_\alpha(i)$ and $f_\beta(i)$

1. Player P_i shares the value $f_\alpha(i)f_\beta(i)$ by choosing a random polynomial $h_i(x)$ of degree t, such that $h_i(0) = f_\alpha(i)f_\beta(i)$. He gives player P_j the value $h_i(j)$ for $1 \leq j \leq 2t+1$.
2. Each player P_j computes his share of $\alpha\beta$ via a random polynomial H, i.e. the value $H(j)$, by locally computing the linear combination $H(j) = \sum_{i=1}^{2t+1} \lambda_i h_i(j)$.

Figure 2: Simplified Multiplication Protocol with honest players

Theorem 2 *Protocol* Simple-Mult *is a secure multiplication protocol in the presence of a passive adversary computationally unbounded.*

In order to tolerate an active adversary there is a need to verify the actions of the players. [BGW88] uses a computationally expensive protocol to do this (which could be combined with Simple-Mult). However, we were able to simplify this protocol as well, and greatly improve its efficiency. The description of our simplification appears in Appendix D.

4 Computations with a Polynomial Time Adversary

In this section we describe how to carry out multiparty computations in the presence of a computationally bounded adversary. It is well known that in this model there exist VSS protocols due to Feldman [Fel87] and Pedersen [Ped91a] which are quite efficient and require limited interaction. We will show that is possible to use these kind of VSS protocols, including our New-VSS, to perform multiparty computations efficiently.

The basic idea is to use a homomorphic commitment (see Section 4.1) to commit to the sharing of the inputs during the VSS. The computation will then follow the [BGW88] paradigm. Additions are computed locally by just summing up the shares of the secret values being added. For multiplication we run a robust version of the simplified multiplication protocol Simple-Mult presented above. But we will use the public commitments over the inputs to enforce correct behavior on the part of the players.

This idea originated in [CCD88] in the information-theoretic model, where such "commitments" were achieved by a second layer of input sharings. In the cryptographic model we use homomorphic commitments to generate the same effect. Some of these techniques have been independently devised by [CDM97], yet they use them in the context of span programs.

In the following sections we will concentrate on the multiplication protocol. Given two secrets α and β shared via some form of VSS, which generated some representation of the secrets, we want to compute a sharing of $\gamma = \alpha\beta$ resulting in the same representation. By representation we mean either the commitment to the coefficients or the commitment to the points of the polynomial. Player P_i holds shares α_i, β_i of α and β (resp.). In order to get a robust version of the multiplication protocol described in Section 3 we need to enforce that P_i shares the product $\alpha_i\beta_i$ via a polynomial of degree t.

4.1 Homomorphic Commitments

The approach we follow requires the usage of *homomorphic commitments*. Denote by $\mathcal{H}(\alpha, \rho)$ a commitment to α with randomness ρ. We shall say that it is a homomorphic commitment if it has the following property: given $A_1 = \mathcal{H}(\alpha_1, \rho_1)$ and $A_2 = \mathcal{H}(\alpha_2, \rho_2)$ it holds for some ρ that: $A_1 \cdot A_2 = \mathcal{H}(\alpha_1 + \alpha_2, \rho)$