EFFICIENT MULTI-PARTY COMPUTATION WITH COLLUSION-DETERRED SECRET SHARING

Zhaohong Wang, Ying Luo, and Sen-ching Cheung

Department of Electrical and Computer Engineering, University of Kentucky

ABSTRACT

Many secure multiparty computation (SMC) protocols use Shamir's Secret Sharing (SSS) scheme as a building block. Unlike other cryptographic SMC techniques such as garbled circuits (GC), SSS requires no data expansion and achieves information theoretic security. A weakness of SSS is the possibility of collusion attacks from participants. In this paper, we propose an evolutionary game-theoretic (EGT) approach to deter collusion in SSS-based protocols. First, we consider the possibility of detecting the leak of secret data caused by collusion, devise an explicit retaliation mechanism, and show that the evolutionary stable strategy of this game is not to collude if the technology to detect the leakage of secret is readily available. Then, we consider the situation in which data-owners are unaware of the leakage and thereby unable to retaliate. Such behaviors are deterred by injecting occasional fake collusion requests, and detected by a censorship scheme that destroys subliminal communication. Comparison results show that our collusion-deterred SSS system significantly outperforms GC, while game simulations confirm the validity of our EGT framework on modeling collusion behaviors.

Index Terms— multi-party computation, collusion, efficiency

1. INTRODUCTION

Privacy protection in distributed computing enables distrusting parties to participate in joint computation without revealing their secret data. The standard approach is to use secure multiparty computation or SMC protocols such as garbled-circuits or homomorphic encryption, which typically leads to significant increase in communication and computation costs [1, 2]. The long security parameters used in these protocols increase the dimension of the ciphertext, making the encrypted-domain processing unsuitable for most practical purposes.

An alternative is to use Information-Theoretic SMC (IT-SMC) protocols with Shamir's Secret Sharing (SSS) as the building block [3]. A wide range of applications of IT-SMC, from image processing [4] to information retrieval [5], have been proposed. Information exchanged between different parties in SSS is statistically independent of the secret data. Also, SSS does not depend on the hardness of specific computational problems. They often admit faster implementations using a smaller finite field for data representations [4]. A major disadvantage is the need to maintain a majority of non-colluding computing parties [3]. Existing solutions rely on either a trusted centralized server [6] or a specially-designed ballot box [7], neither of which are ideal for large-scale IT-SMC deployment.

In this paper, we experimentally validate the complexity advantage of IT-SMC over GC, and address the collusion problem using a game-theoretical framework. Our proposed solution differs from existing solutions in two areas. First, we consider the possibility of detecting the leakage of secret data and devise an explicit retaliation mechanism in an evolutionary game-theoretic context such that players are forced to be honest. Second, we consider the case of "undetectable" theft where data-owners are unaware of the leakage. Such behaviors are deterred by injecting occasional fake collusion requests, and detected by a censorship scheme. To the best of our knowledge, this is the first work to use evolutionary game in deterring collusion in IT-SMC and to consider both detectable and undetectable collusion.

The rest of the paper is organized as follows: Section 2 describes collusion attacks and motivates our solutions. Section 3 reviews existing works for anti-collusion in IT-SMC. Sections 4 to Section 6 describe three different types of attacks and the solutions. We provide experimental results in Section 7, and conclude the paper in Section 8.

2. PROBLEM STATEMENT AND SOLUTION OVERVIEW

We consider two types of entities in a distributed computation protocol: the computing platform and the platform users. Denote a pair of users as U (a customer) and V (a software vendor), each with respective secrets x and y that are used for a joint computation. The actual computation is outsourced to the computing platform P which consists of a large number of autonomous computing agents. We assume a covert adversarial model in that every participant is interested in stealing other's secrets without getting caught. This model excludes arbitrary malicious behaviors like abrupt termination but opens door to possible collusion attacks among multiple conspiring participants. We will focus only on attacks on the two operations: addition and multiplication. These operations are universal in building any arithmetic and logical procedures, and are sufficient in illustrate the problem and our proposed solution. In Section 7, we will demonstrate the performance of more sophisticated algorithms using these operations as building blocks.

In SSS, a secret x from a finite field F_m, where m is the size of the field, can be decomposed into n shares $\{[x]_i^t, i = 1, \ldots, n\}$ such that any subset with less than the threshold $t \leq \lceil n/2 \rceil$ shares contains no information about x. Equations 1 and 2 describe the share generation and secret reconstruction processes respectively.

$$[x]_i^t \triangleq \sum_{j=1}^{t-1} \alpha_j i^j + x \bmod m, \tag{1}$$

$$x = \sum_{i \in K} \gamma_i [x]_i^t \bmod m, \tag{2}$$

α_j's are random numbers known only to the share generator, $\gamma_i \triangleq$

This work was supported in part by the National Science Foundation under Grant 1018241.

$\prod_{1\le j\le n, j\neq i} \frac{-k_j}{k_i - k_j}$ are reconstruction coefficients, and K is any subset of $\{1, \ldots, n\}$ with at least t elements.

Suppose the goal is to compute $x + y$ where x and y are secret data from U and V. U and V can decompose x and y into n secret shares and send them to different agents. Each agent can homomorphically compute the secret shares of $x + y$ without any communication:

$$[x + y \bmod m]_i^t = [x]_i^t + [y]_i^t \bmod m. \quad (3)$$

This simple operation is prone to two possible collusion attacks:

A1: Side-channels among agents for collusion

If an adversary controls t or more computing agents involved in the computation, they can exchange their secret shares through this side channel to reconstruct the secret numbers x and y. Protection against side-channel attacks is beyond the scope of this paper and we will simply assume that no such side channels exist.

A2: Collusion between agents and U or V

We will focus on agent's collusion with U, as the case for V is identical. As each agent possesses secret shares from both U and V, it is possible for U to collude with t or more agents to reconstruct y of V. No changes in infrastructure can block such collusion as it is necessary for U to communicate with the agents. In Section 4, we study the mechanism to deter such an attack by evolutionary games.

Let us now consider multiplication of x and y. The constant term of the product of the two secret polynomials is indeed xy, but the degree of the product polynomial increases to $2t - 2$. Repeated applications of such operations will eventually arrive at a threshold larger than n so the final result cannot be reconstructed. This problem can be resolved by either increasing the number of agents so that it is large enough to cover all operations, or by applying a "renormalization" procedure to reduce the threshold back to t [4] where each agent breaks its product share into n separate shares, and sends one share to each of the corresponding agents. The final share at each agent is computed as

$$[xy \bmod m]_i^t = \sum_{j=1}^{n} \gamma_j [[xy \bmod m]_j^{2t-1}]_i^t \bmod m. \quad (4)$$

No secret is leaked through the renormalization process. It can be shown that renormalization requires less communication than adding more agents [8]. However, renormalization requires agents to communicate directly with each other, leading to the last collusion attack.

A3: Collusion attack by computing agents

The direct communication among agents enables them to exchange secret shares. The difference between **A1** and **A3** is that the coalition of agents in **A3** forms after the assignment of agents to U and V. As such, it is possible for U and V to thwart this collusion by inspecting the communication among agents, which is described in Section 6.

3. RELATED WORK

Various anti-collusion schemes have been studied for SMC applications. The scheme in [7] requires special hardware called a verifiably secure device or VSD. In [6], the mediated multiparty computation (MMPC) achieves the collusion-deterrence with computational security by means of a mediator involving in a two-party secure function evaluation (SFE) with each party, where all parties are isolated without any side channels. In contrast, our solution without any special hardware is far more scalable than MMPC for data-intensive SSS-based signal processing applications.

In this paper, we use evolutionary game theory (EGT) to model collusion behaviors in distributed computing. EGT considers a population of decision makers in which the frequency of a particular strategy can change over time in response to the decisions made by all individuals. In distributed computing where a large amount of users interact, a player can learn from mistakes from prior interactions to adapt his/her strategy over time. This dynamics is exactly what evolution suggests. EGT has been successfully applied in solving various networking problems including peer-to-peer streaming [9]. In SMC, we consider the emergence of a sustainable strategy called evolutionarily stable strategy or ESS [10], chosen between staying honest and colluding, through repeated interactions. Our goals are to derive the conditions under which staying honest is the ESS. The security of our approach is guaranteed by the underlying SSS building blocks.

4. CUSTOMER-VENDOR GAME

Before starting the joint computation, U and V should understand that they are bounded by contract not to collude with agents in stealing each other's secret. If V finds out U trying to steal V's secret, U would be liable to pay for damages. Such judgement, however, can only be rendered by an appropriate authority after possibly a long proceeding to evaluate all the evidence. The cost and effort of collecting evidence and going through the proceeding makes such this "retaliation" outcome the most undesirable one for all parties. To formulate the possible cheating behaviors in game, we need to rank all outcomes and map them to appropriate payoff functions [10].

The rankings used in this paper are described in Table 1 and can be understood as follows: as postulated before, the lowest rank outcome is retaliation brought on by the cheating strategy of either U or V. The second lowest-ranked outcome is when both have cheated but neither retaliates – even though both U and V steal each other's secrets without getting caught, the fact that they both cheat would imply that they have wasted resources colluding with the agents in stealing something that is of little value. If only U cheats and gets away with it, U will have the highest-rank (5) outcome. As for V, we give a rank of 3 for two reasons: (1) V successfully carries out the task and gets compensated; and (2) V does not retaliate because either (a) he is unaware of the theft as he does not put in a significant effort in tracking any leakage of his secret, or (b) the cost of retaliation is higher than the cost of his secret. Either reason implies that the loss of the secret may not be too significant to V. The situation is identical if we switch U and V. Finally, we assign the second highest rank to both U and V when they complete the task faithfully.

Table 1. Ranking of different outcomes in customer-vendor game

Strategies	Retaliate?	U Rank	V Rank
Either or both cheats	Y	1	1
Both cheat	N	2	2
U cheats only	N	5	3
V cheats only	N	3	5
No one cheats	N	4	4

We denote the normalized payoff values for these outcomes as $0 = p_0 < p_1 < p_2 < p_3 < p_4 = 1$. Let q be the "non-retaliate" probability for both U and V conditioned on the other's cheating behavior. The value $q = 1$ means that no one retaliates while $q = 0$ means that one always retaliates if his/her secret is stolen. The *customer-vendor* game can now be described in Table 2. The two-tuple in each entry indicates the average payoffs of U and V when adopting the row and column strategies respectively. In the context

of population game, cheating would be an ESS if (a) $q^2p_1 > qp_2$ or (b) $q^2p_1 = qp_2$ and $q > p_3$. As $0 \leq p_1, q \leq 1$, neither conditions are valid and cheating can never be an ESS. Honesty would be an ESS if (a) $p_3 > q$ or (b) $p_3 = q$ and $qp_2 > q^2p_1$. As $qp_2 > q^2p_1$ is always true, we have the following conclusion: **Honesty is an ESS for both U and V if $\mathbf{p_3 \geq q}$.** When the theft is *undetectable*, i.e. $p_3 < q$, it can be shown that the following mixed strategy constitutes an ESS [11]:

$$h_u = h_v = \frac{1}{\frac{q-p_3}{q(p_2-qp_1)} + 1} \tag{5}$$

where h_u and h_v are the honest fraction of U and V respectively. Unfortunately, this situation will undoubtedly occur in real life. It is thus important to incorporate additional mechanisms to deter cheating behaviors.

Table 2. Customer-Vendor Game

		V: Honest	V: Cheat
U	Honest	(p_3, p_3)	$(1-q)(p_0,p_0)$ $+q(p_2,p_4)$ $=(qp_2, q)$
	Cheat	$(1-q)(p_0,p_0)$ $+q(p_4,p_2)$ $=(q, qp_2)$	$(1-q^2)(p_0,p_0)$ $+q^2(p_1,p_1)$ $=(q^2p_1, q^2p_1)$

5. USER-AGENT GAME

For U to be successful in stealing V's secret, U must be able to convince t or more agents to collude with him/her. A collusion attack can thus be avoided if the agents refuse to collude. To deter agents from colluding with users, we introduce honest undercover users (police) that attempt to collude with agents. A cheating agent who is reported by either a police user or a honest user will be paid nothing and banned from the system. This worst outcome is denoted by p_0. Let λ be the conditional probability of encountering police given a colluding request from the user. The payoff matrix for the *user-agent* game is given in Table 3.

Table 3. User-Agent Game

		A: Honest	A: Cheat
U	Honest	(p_1, p_1)	(p_0, p_0)
	Cheat	$\lambda \cdot (p_1,p_1)+$ $(1-\lambda)(p_1,p_1)$ $=(p_1, p_1)$	$\lambda \cdot (p_0,p_0)+$ $(1-\lambda)(p_2,p_2)$ $=(1-\lambda, 1-\lambda)$

The normalized payoffs are represented as $0 = p_0 < p_1 < p_2 = 1$, and are assumed to be the same for both user and agents for simplicity. There are two pairs of Nash Equilibriums (NE) in this game. When $p_1 > 1-\lambda$, the agent staying honest is dominant while no dominant strategy exists for the user. When $p_1 < 1-\lambda$, the NE becomes cheating for both the user and the agent. Such an unfortunate situation will occur when there are not enough police or the payoff for an honest agent is significantly smaller than collusion. There is no NE for the case when $p_1 = 1-\lambda$.

6. COLLUSION FREENESS AMONG AGENTS

For the attack **A3**, we propose a censorship scheme that relies on U and V to detect subliminal communication among agents. This scheme requires routing all agents' traffic through U and V. While this is similar to the mediator solution in [6], our scheme is of much lower complexity as it is needed for renormalization only. Our scheme will introduce $2n$ more invocations of communication in each renormalization compared to that without any collusion-deterrence, and two more invocations of communication in each reconstruction of an intermediate result.

Assume the the threshold of an intermediate result has reached $d > \lceil \frac{n}{2} \rceil$ and a renormlization is needed. Denote the intermediate answer as x and the share at agent P_i as $[x]_i^d$ for $i = 1, 2 \ldots, n$. $[x]_i^d$ for $i = 1, \ldots, \lceil \frac{n}{2} \rceil$ are sent to U and the rest are sent to V. Afterwards, U and V generate the renormalized shares as follows:

$$[[x]_i^d]_j^t = \sum_{k=1}^{t-1} \alpha_k j^k + [x]_i^d \bmod m, \tag{6}$$

where α_k are random numbers unknown to any agents. $[[x]_i^d]_j^t$ are then sent to agent P_j for $i, j = 1, 2, \ldots, n$. To complete the renormalization process, P_j can compute $[x]_j^t$ based on Equation (4).

Security Proof:
As $d > \lceil \frac{n}{2} \rceil$, neither U nor V can learn anything about the intermediate secret x. In order to establish subliminal communication between agents P_i and P_j, the renormalized share must be replaced by a certain pattern acting as a preamble for collusion. However, the received shares $[[x]_i^d]_j^t$ are no different to P_j than random numbers and thereby completely destroy any preambles. As such, no subliminal communication can be established among agents.

7. EXPERIMENTS

In this section, we first present a comparison in computation efficiency between our Collusion-Deterred SSS (CD-SSS) and Garbled Circuits (GC). Then, we simulate how different strategies might evolve under different conditions in the customer-vendor game and user-agent game.

7.1. Computational Efficiency of CD-SSS versus GC

We first test the hypothesis that our CD-SSS system provides a much more computationally-efficient SMC system than garbled-circuit (GC). GC is a 2-party SMC theme. To have a fair comparison with our proposed system, the GC evaluation is performed at each of the agents in parallel based on secret inputs provided by the customer and vendor. Our GC implementation is based on the optimized Java library as described in [12].

We compare the performance of CD-SSS and GC on addition, scalar multiplication, and comparison. While the first two are straightforward, the comparison protocol is complex and our implementation is based on the optimized algorithm described in [13]. Here we briefly review the procedure: suppose x and y are two $(k-1)$-bit numbers to be compared and $d = x - y$. Notice that $d < 0$ implies $\lfloor d/2^{k-1} \rfloor = -1$ while $d \geq 0$ implies $\lfloor d/2^{k-1} \rfloor = 0$. The comparison protocol thus comprises of a truncation protocol in computing $\lfloor d/2^{k-1} \rfloor$, bitwise comparison of a secret number d with 0, and reconstruction of the output results. Each of the above three subroutines involves multiple shared random number generations, multiplications, renormalization, and additions.

In the proposed CD-SSS, the customer and vendor are responsible for renormalization and reconstructions, while the agents perform all the remaining computation. We adopt a number of strategies to expedite the calculations. First, all shared random numbers are pre-generated and distributed among the agents. Second, as communication and synchronization are needed after comparing each bit, we amortize the measurements over a large number of comparison

operations in a bitwise fashion so as to minimize the communication overhead. To promote reproducible research, we have made our CD-SSS implementation publicly available at our website [1].

Our test bed consists of five different machines. The customer and vendor processes are run on two WinTel machines on the same 100Mbps LAN, each with Intel Core Quad CPU at 3.00GHz with 4 GB memory. The three agents are placed at three nodes of a Dell C6220 Server in a separate LAN where each node is a dual Intel E5-2670 8 Core running at 2.6 GHz with 64 GB memory. The amortized results over $100,000$ operations are shown in Table 4. All the operands are 8-bit signed integers and the modulo field $\mathbb{Z}_q$ used in CD-SSS is based on $q = 65539$. GC has the security parameter of 80 bits.

Table 4. CPU and Bandwidth per operation for CD-SSS and GC

		CD-SSS		GC
		Agent	User	
Add	CPU (μs)	0.300	0.910	1982
	Comm. (Bytes)	32		928
Scale	CPU (μs)	0.300	0.910	14464
	Comm. (Bytes)	32		7738
Compare	CPU (μs)	52	14	1448
	Comm. (Bytes)	126		726
	Num. Network calls	22		-
	Wall time (μs)	159.5	82.3	-

We measure the CPU time of both the agent (parallel time) and the users and the communication bandwidth in CD-SSS, while combining those of the agent and user together in GC. For comparison in CD-SSS, we also count the number of network invocations, and the whole process' wall time. These are not calculated in GC because it is not real-time. Note that GC is more efficient in bitwise comparison than addition and scale while the reverse is true for CD-SSS. Overall CD-SSS is $20 - 10,000$ times more efficient than GC.

7.2. Game simulation of cheating behaviors

We use the replicator dynamic (RD) to simulate the evolution from a given population profile under different conditions. Under RD, the growth rate of the agents using each strategy is proportional to the excess of the strategy's payoff over the average payoff [10]. Our implementations are based on the GameBug simulator [14].

We first illustrate how the system evolves over time for the Customer-Vendor Game. Each customer in the system is randomly matched with a vendor from the same population for cooperations. Recall that q is the non-retaliation probability and p_i is the i-th ranked payoff. At the beginning of the simulation, 90% of the populations are honest. We first test the scenario of $q \leq p_3$. The initial population profile evolves very quickly toward the pure-honesty ESS as depicted in Fig. 1. In sharp contrast, $q > p_3$ leads to a mixed ESS with $h_u = h_v = 0.8$. Fig. 2 shows this evolution in the population whose profile gradually converges to the mixed ESS as marked by the black solid line.

Next, we simulate the User-Agent game. Consider the case when $p_1 > 1 - \lambda$. Setting 50% of the users and agents as honest initially, Fig. 3 shows that agents evolve to honesty while users stay at a mixed strategy over time as predicted by the non-strict NE. Second, for the case of $p_1 < 1 - \lambda$, Fig. 4 shows that the same initial population composition of half cheating and half honest is gradually taken over by cheating which is an ESS.

[1] http://vis.uky.edu/nsf-privacy/information-theoretic-smc/

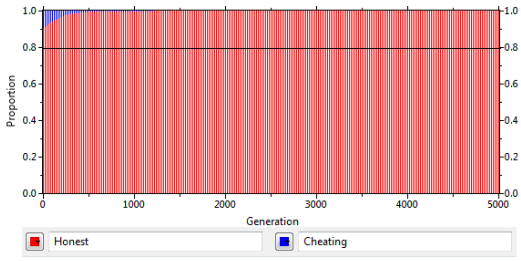

Fig. 1. Emergent Behavior in the Customer-Vendor game when $q < p_3$

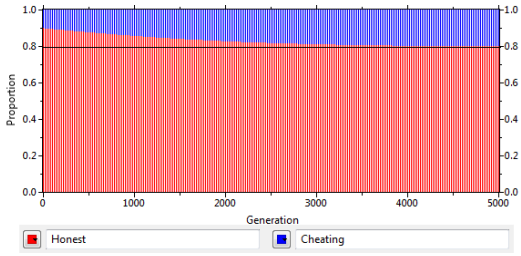

Fig. 2. Emergent Behavior in the Customer-Vendor game when $q > p_3$

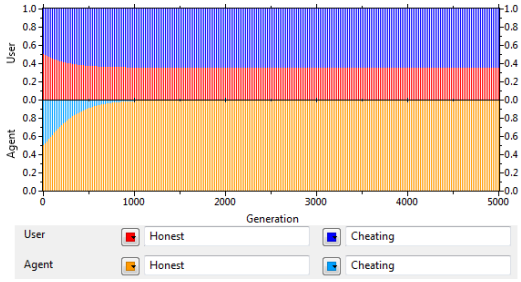

Fig. 3. Emergent Behavior in the User-Agent game when $p_1 > 1 - \lambda$

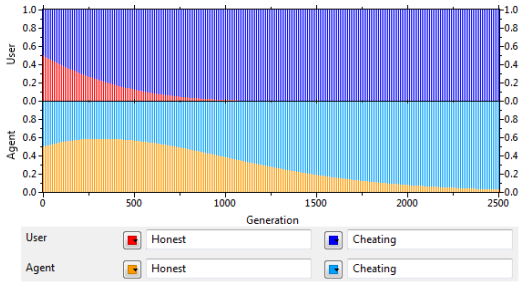

Fig. 4. Emergent Behavior in the User-Agent game when $p_1 < 1 - \lambda$

8. CONCLUSIONS

In this paper, we have demonstrated the efficiency of SSS-based IT-SMC system over the traditional GC, and have proposed novel EGT-based technique in deterring collusion attacks against such a system. Further investigations are needed to confirm whether the EGT model accurately reflects practical applications and to extend to cope with the coexistence of games with a wide range of privacy values.

9. REFERENCES

[1] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, et al., "Secure multiparty computation goes live," in *Financial Cryptography and Data Security*, pp. 325–343. Springer, 2009.

[2] Ronald Cramer and Ivan Damgård, "Multiparty computation, an introduction," in *Contemporary cryptology*, pp. 41–87. Springer, 2005.

[3] Josh Cohen Benaloh, "Secret sharing homomorphisms: Keeping shares of a secret secret," in *Advances in Cryptology-CRYPTO86*. Springer, 1987, pp. 251–260.

[4] Sayed M SaghaianNejadEsfahani, Ying Luo, and Sen-ching S Cheung, "Privacy protected image denoising with secret shares," in *Image Processing (ICIP), 2012 19th IEEE International Conference on*. IEEE, 2012, pp. 253–256.

[5] Casey Devet, Ian Goldberg, and Nadia Heninger, "Optimally robust private information retrieval," in *21st USENIX Security Symposium*, 2012.

[6] Joël Alwen, Jonathan Katz, Ueli Maurer, and Vassilis Zikas, "Collusion-preserving computation," in *Advances in Cryptology–CRYPTO 2012*, pp. 124–143. Springer, 2012.

[7] Sergei Izmalkov, Matt Lepinski, and Silvio Micali, "Verifiably secure devices," in *Theory of Cryptography*, pp. 273–301. Springer, 2008.

[8] Y. Luo, S.M. Esfahani, and S.-C. Cheung, "Privacy-protected image processing with secret shares," *IEEE Transactions on Information Forensics and Secuity*, 2013, In preparation.

[9] Yan Chen, Beibei Wang, W. Sabrina Lin, Yongle Wu, and K. J. Ray Liu, "Cooperative peer-to-peer streaming: An evolutionary game-theoretic approach," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 20, no. 10, pp. 1346–1357, 2010.

[10] James N Webb, *Game theory: decisions, interaction and Evolution*, Springerverlag London Limited, 2006.

[11] Z. Wang and S.-C. Cheung, "Collusion deterrence in secure multiparty computation," *IEEE Transactions on Information Forensics and Secuity*, 2014, In preparation.

[12] Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider, "Improved garbled circuit building blocks and applications to auctions and computing minima," in *Cryptology and Network Security*, pp. 1–20. Springer, 2009.

[13] Octavian Catrina and Sebastiaan De Hoogh, "Improved primitives for secure multiparty integer computation," in *Security and Cryptography for Networks*, pp. 182–199. Springer, 2010.

[14] Robert Wyttenbach, "Gamebug software," .