Optimizations for Secure Multiparty Computation Protocols

Thilo Linke, Christopher Harth-Kitzerow*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: thilo.linke@tum.de, christopher.harth-kitzerow@outlook.de

Abstract—**The Beaver-Micali-Rogaway protocol describes a method for securely computing functions with any number of participating parties, that builds on the principles of Yao's Garbled Circuits protocol for two participants. Its key advantage over similar protocols is that it only requires a fixed constant amount of communication rounds to build the garbled circuits. Moreover, it is possible to apply the FreeXOR optimization technique to the protocol in order to simplify the evaluation of XOR gates of the garbled circuits and thereby improve overall runtime.**

Index Terms—**secure multiparty computation, bmr protocol, freexor optimization technique**

1. Introduction

A simple practical scenario that requires the usage of a Secure Multiparty Computation (SMC) protocol is to carry out a private vote. The concrete purpose of SMC is to provide protocols that keep computation input data from participants private, while not requiring any trusted third parties [1].

Yao's Garbled Circuits protocol (YGC) for two parties, introduced in 1983 by A. C. Yao [2], started the long evolution of SMC protocols [1]. The GMW protocol from O. Goldreich et al. [3] was one of the first to enable computations with any number of participants.

This paper presents the Beaver-Micali-Rogaway protocol (BMR), introduced in 1990 by Donald Beaver, Silvio Micali and Phillip Rogaway, which generalizes YGC's concepts in order to enable any number of participants [4], [1]. Its constant number of required communication rounds for building its computation structure makes it especially attractive for scenarios with high network latency, like computation over the internet [5].

After providing an overview of the original BMR protocol, we describe an optimization introduced by A. Ben-Efraim et al. in 2016 [5], who apply the FreeXOR technique to evaluate XOR gates of garbled circuits more efficiently.

2. The BMR Secure Multiparty Computation Protocol

The BMR protocol adapts the garbled circuit concept from YGC, which cryptographically guarantees that input values are kept secret. Before actually describing the protocol, Section 2.1 serves as an introduction to this concept. The following description is based on the circuit definition from the original version of the BMR protocol from [4]. We use symbols similar to those from the BMR protocol adaptation from [5].

2.1. Garbled Circuits

A garbled circuit consists of wires, signals and gates. These building blocks can be used to construct any arbitrary computable function, like in an ordinary Boolean circuit.

2.1.1. Wires. The wires can carry one of two signals and connect the gates. Each party initially holds some data for the circuit input wires and the combined input from all parties is needed to execute the circuit. The values obtainable from the circuit output wires are the result of the computation.

2.1.2. Signals. Garbled circuits encrypt the Boolean signal values to hide them from the participating parties and ensure input secrecy.

Like in the YGC protocol, wire ω does not carry signals 0 or 1, but the secret random binary strings $k_{\omega,0}$ or $k_{\omega,1}$, which get collaboratively generated by the parties. The key modification of Yao's two-party method is that each party i of the n computing parties possesses its own private share of the strings in form of substrings $k^i_{\omega,0}$ and $k^i_{\omega,1}$ of the length of the cryptographic security parameter κ [1]. The two private signals for each wire therefore are

$$k_{\omega,\tau} = k^1_{\omega,\tau} \cdots k^n_{\omega,\tau} \text{ for } \tau \in \{0,1\}. \tag{1}$$

During circuit creation, each party additionally generates a secret share λ^i_ω of a random *permutation bit* λ_ω for each wire ω. The real meaning of a signal string $k_{\omega,\tau}$ of the circuit is then defined as Boolean value

$$\lambda_\omega \oplus \tau = \lambda^1_\omega \oplus \cdots \oplus \lambda^n_\omega \oplus \tau \tag{2}$$

and because nobody knows the value of λ_ω, the hidden Boolean signals are effectively concealed. This is the mechanism that actually enables input privacy. [5]

2.1.3. Gates. A gate g with left and right input wires α and β, as well as output wire γ calculates an arbitrary Boolean function on its inputs. Since the input and output signals are random strings that conceal their underlying values, the gates have to work with a specific mechanism to enable output computation.

Each gate g holds a table of four $n \cdot \kappa$ bit long *gate label* strings $X^{a,b}_g$ for each possible combination of inputs

$k_{\alpha,a}$ and $k_{\beta,b}$ [4]. Informally, the labels are associated with the signals via a truth table. More formally:

$$((\alpha \text{ carries } k_{\alpha,a}) \wedge (\beta \text{ carries } k_{\beta,b})) \longleftrightarrow X_g^{a,b} \tag{3}$$

The core of the gate label creation process is to seed a pseudorandom generator with the gate input signal strings and then to mask the gate output signal strings with the result. The encrypted output signals are then used as gate labels. Details on how this can be achieved and how circuit evaluators use the labels to compute the output of a gate can be found in the following section.

2.2. The Protocol

Overall, the BMR protocol computation process is commonly divided into two phases. The first phase utilizes SMC between the n participants to generate the garbled circuit and inputs, while in the second phase, each party executes the built circuit on their own to obtain the result.

This section explains the structure and required steps of the original protocol from [4] in order to provide a basic foundation. There exist many subsequent descriptions, like from [5], [6] or [7], that additionally apply various modifications. We mention two of those enhancements in Section 2.3.

2.2.1. Phase 1.

The protocol starts by constructing the garbled circuit and its garbled inputs. Any secure protocol like BGW or GMW can be used for steps that require SMC [1], [5], [6].

Signal Creation. *Secret sharing* is used to generate random bit strings. Essentially, each party i privately generates random bit strings s^i that are called shares. The actual value is then defined as

$$s = \bigoplus\nolimits_{i=1}^{n} s^i. \tag{4}$$

This means that the resulting value remains secret, unless someone possesses all shares at once. [4]
The two steps to compute the signals are:

1) The parties generate the permutation bits λ_ω for each wire ω by creating the private λ_ω^i shares [4].
2) They additionally need to create the secret random signal strings $k_{\omega,\tau}$ for $\tau \in \{0,1\}$ [4]. After this step, each party holds private substrings $k_{\omega,\tau}^i$ for each wire ω.

Label Creation. As we mentioned in Section 2.1.3, each gate will hold a table of four strings, called gate labels. The original BMR design additionally associates all wires of the garbled circuit with public labels. Because the creation of the labels can be accomplished in parallel and the required communication is independent of the size of the circuit, it is very efficient [1].

To generate the labels, party i locally uses a pseudorandom generator G, which takes each of their previously obtained private signal substrings of length κ for wire ω as input and transforms them to pseudorandom strings of length $\kappa + 2 \cdot n \cdot \kappa$. To be precise,

$$G(k_{\omega,\tau}^i) = x_{\omega,\tau}^i y_{\omega,\tau}^i z_{\omega,\tau}^i, \tag{5}$$

where $|x_{\omega,\tau}^i| = \kappa$, $|y_{\omega,\tau}^i| = n \cdot \kappa$ and $|z_{\omega,\tau}^i| = n \cdot \kappa$. [4]

Each party has to prove via zero-knowledge-proofs to the other parties that they truthfully calculated these strings, as a measure to rule out malicious intent [4].

The produced strings are processed as follows:

1) The $x_{\omega,\tau}^i$ strings for $\tau \in \{0,1\}$ are used to form public *wire labels* $x_{\omega,\tau} = x_{\omega,\tau}^1 \cdots x_{\omega,\tau}^n$ [4]. If a circuit evaluator obtains signal $k_{\omega,a}$ for a wire ω during the second phase, they can use G to calculate the same wire label that the signal produced previously in phase one. The knowledge about which of the two publicly known wire labels they obtain from this allows them to choose the correct gate label to proceed (see Section 2.2.2).
2) The $y_{\omega,\tau}^i$ and $z_{\omega,\tau}^i$ strings for $\tau \in \{0,1\}$ are used for collaborative gate label creation and remain private [4]. Each of these labels encrypts one output signal string of a gate. Because G used the input signals as seeds, they are also the keys to decrypt the output signals. The association between input signals and gate labels is shown in (3).
   For example, if a circuit evaluator holds signals $k_{\alpha,0\oplus\lambda_\alpha}$ and $k_{\beta,1\oplus\lambda_\beta}$ for left and right input wires α and β at AND gate g, the associated gate label for the signals should encrypt signal $k_{\gamma,0\oplus\lambda_\gamma}$ for output wire γ. The following equations, adapted from [4], that get securely and collaboratively evaluated by the parties using SMC, ensure this:

$$X_g^{0,0} = \bigoplus\nolimits_{i=1}^{n} (y_{\alpha,0}^i \oplus y_{\beta,0}^i) \oplus k_{\gamma,f_g(\lambda_\alpha,\lambda_\beta)\oplus\lambda_\gamma},$$
$$X_g^{0,1} = \bigoplus\nolimits_{i=1}^{n} (z_{\alpha,0}^i \oplus y_{\beta,1}^i) \oplus k_{\gamma,f_g(\lambda_\alpha,\overline{\lambda_\beta})\oplus\lambda_\gamma},$$
$$X_g^{1,0} = \bigoplus\nolimits_{i=1}^{n} (y_{\alpha,1}^i \oplus z_{\beta,0}^i) \oplus k_{\gamma,f_g(\overline{\lambda_\alpha},\lambda_\beta)\oplus\lambda_\gamma},$$
$$X_g^{1,1} = \bigoplus\nolimits_{i=1}^{n} (z_{\alpha,1}^i \oplus z_{\beta,1}^i) \oplus k_{\gamma,f_g(\overline{\lambda_\alpha},\overline{\lambda_\beta})\oplus\lambda_\gamma},$$

   where $f_g(\cdot,\cdot)$ is the Boolean gate function, which would be AND in the previous example. Section 2.2.2 explains how the masked output signal can be obtained from a gate label, if the gate input is known to an evaluator.

Garbled Input Creation. By using SMC, the parties decide which of the two signals for each input wire ω gets chosen as input for the circuit. To do this, the party who owns input bit b_ω for wire ω has to secretly share it with the other participants. The input, in combination with the already secretly shared permutation bit λ_ω and signal strings $k_{\omega,0}$ and $k_{\omega,1}$, are the required information to choose the correct signal $k_{\omega,b_\omega\oplus\lambda_\omega}$ as garbled input. [4]

2.2.2. Phase 2.

At first, all wire labels, gate labels, garbled input signals, as well as the permutation bits of the circuit output wires are sent to all participants [4]. After this, they can independently evaluate the circuit and obtain the calculation result at the output wires.

When a participant knows left input $k_{\alpha,a}$ and right input $k_{\beta,b}$ for a gate g, they can calculate $x_{\alpha,a}$ and $x_{\beta,b}$ by using G and compare the result with the public wire labels. The values a and b associated with the labels are known and thereby the evaluator now knows those same values a and b of the signals it holds. To obtain the gate output for output wire γ, the party has to solve the previous equations for calculating the gate labels for the output signal

$$k_{\gamma,c} = \begin{cases} \bigoplus_{i=1}^{n}(y^i_{\alpha,0} \oplus y^i_{\beta,0}) \oplus X_g^{a,b}, \text{ if } a=0, b=0 \\ \bigoplus_{i=1}^{n}(z^i_{\alpha,0} \oplus y^i_{\beta,1}) \oplus X_g^{a,b}, \text{ if } a=0, b=1 \\ \bigoplus_{i=1}^{n}(y^i_{\alpha,1} \oplus z^i_{\beta,0}) \oplus X_g^{a,b}, \text{ if } a=1, b=0 \\ \bigoplus_{i=1}^{n}(z^i_{\alpha,1} \oplus z^i_{\beta,1}) \oplus X_g^{a,b}, \text{ if } a=1, b=1 \end{cases}$$

by using G again with the input signals. [4]

When an evaluator arrives at a circuit output wire ω, they can decrypt its signal $k_{\omega,\tau}$ by using the public permutation bit λ_ω to calculate $\tau \oplus \lambda_\omega$, in order to reverse (2). [4]

2.3. Protocol Improvements

- Wire labels can be omitted, as has been implemented by [5], because party i can simply compare $k^i_{\omega,a}$ with the shares $k^i_{\omega,0}$ and $k^i_{\omega,1}$ it owns and thus decide which of the signals it holds.
- Recent implementations of the protocol show that there exist more efficient alternatives for the earlier mentioned expensive zero-knowledge-proofs, without sacrificing any security [7].

2.4. Security of the BMR protocol

The BMR protocol is secure as long as honest parties, i.e., those who supply correct values for the computation, are in the majority [4].

Security against honest-but-curious adversaries, i.e., those who supply valid values but try to obtain private information, is guaranteed as long as at least one participant remains uncorrupted [1].

Since BMR is a cryptographic protocol, the security concept relies upon the assumption that adversaries can only act in polynomial time [4].

3. Applying the FreeXOR Optimization Technique to the BMR Protocol

In 2009, V. Kolesnikov and T. Schneider introduced the FreeXOR optimization technique for the two-party YGC protocol [8] and subsequent adaptation of it for the BMR protocol happened in [5].

FreeXOR essentially trivializes the construction and evaluation of XOR gates and thus can dramatically improve the runtime of both phases of the protocol. To accomplish this, garbled signal and gate layouts have to be modified. In this section, we are explaining how [5] did this.

The BMR FreeXOR adaptation from [5] only uses XOR and AND gates for its circuits. Here, AND gates are the only gates that still require gate labels and since XOR gates are negligible, circuits using as many XOR instead of other gates as possible are preferable.

Like many adaptations of the original BMR protocol do, that from [5] omits the usage of wire labels (see Section 2.3).

Note that in the following description of the modifications to the garbled circuit design, the usage of secretly shared values, like permutation bits, implies that during BMR protocol execution, the actual equations get securely evaluated on the shared values using SMC to retain secrecy.

3.1. Signal Modifications

The key idea that enables FreeXOR is understanding that making the values of the signal pairs of a wire dependent on each other does not invalidate security [8].

The signal share pair of party i for gate wire ω, *that is not an output wire of a XOR gate*, is created as

$$k^i_{\omega,1} = k^i_{\omega,0} \oplus R^i, \quad (6)$$

where R^i is of length κ. Here, $k^i_{\omega,0}$ remains random. R^i is party i's substring of the global value $R = R^i \cdots R^n$, called the *difference string*, which is created and secretly shared between parties the same way as the signals. The computation process of permutation bit λ_ω remains unchanged (see Section 2.2.1). [5]

The creation of *output wire signals for XOR gates* requires special care. Core of the optimization technique is that the computation of the output signal of a XOR gate does not require any gate labels [8].

Let an XOR gate g have input wires α and β, as well as output wire γ. Assume that wire α carries signal $k_{\alpha,u\oplus\lambda_\alpha}$ and wire β carries signal $k_{\beta,v\oplus\lambda_\beta}$, where u and v are the hidden semantics of the signals. The public output signal semantics for gate g is defined as the XOR of the input semantics. To enable this, permutation bit λ_γ is not random anymore, instead it is simply set as $\lambda_\gamma = \lambda_\alpha \oplus \lambda_\beta$ [5]. This permits that during circuit evaluation in phase 2, public output semantics can be obtained by calculating

$$\begin{aligned}(u \oplus \lambda_\alpha) \oplus (v \oplus \lambda_\beta) &= (u \oplus v) \oplus (\lambda_\alpha \oplus \lambda_\beta) \quad (7)\\ &= (u \oplus v) \oplus \lambda_\gamma. \quad (8)\end{aligned}$$

As has been described in Section 2.2.2, values $u \oplus \lambda_\alpha$ and $v \oplus \lambda_\beta$ are known to an evaluator of the garbled circuit, if they hold signals $k_{\alpha,u\oplus\lambda_\alpha}$ and $k_{\beta,v\oplus\lambda_\beta}$ for the input wires of a gate.

Additionally, instead of being random, the *output signal pair* gets computed as

$$k_{\gamma,0} = k_{\alpha,0} \oplus k_{\beta,0} \text{ and } k_{\gamma,1} = k_{\gamma,0} \oplus R \quad (9)$$

respectively, where R is the aforementioned difference string [5]. The purpose of this is described in Section 3.2.2.

3.2. Gate Modifications

Since output signal creation differs between gate types, the gate design has to be modified depending on the gate type as well.

3.2.1. AND Gates. The computation of gate labels for AND gate g with input wires α and β, as well as output wire γ, happens according to (11) [5]. It essentially adapts the familiar gate label definition from Section 2.2.1 to the new signal design. Note that output signal $k_{\gamma,0}$ gets chosen for all labels and let $a = u \oplus \lambda_\alpha$ and $b = v \oplus \lambda_\beta$ be the public semantics of the input signals.

$$m = R \cdot (((a \oplus \lambda_\alpha) \cdot (b \oplus \lambda_\beta)) \oplus \lambda_\gamma) \quad (10)$$

$$X_g^{a,b} = F_{k_{\alpha,a},k_{\beta,b}} \oplus k_{\gamma,0} \oplus m \quad (11)$$

Here, $F_{k_{\alpha,a},k_{\beta,b}}$ is the processed output of a pseudorandom function. Its definition from [5] does differ from that of the pseudorandom generator in Section 2.2.1, but the purpose remains the same.

The calculation of m in (10) is the result of the fact that

$$u \wedge v = (a \oplus \lambda_\alpha) \wedge (b \oplus \lambda_\beta) \tag{12}$$

is equal to the multiplication of the bit values [5]. Now, two cases depending on parameter λ_γ have to be considered [5].

1) Let $\lambda_\gamma = 0$. If the result of (12) is 0, (10) yields $m = 0$ as well and the gate label masks $k_{\gamma,0}$. Otherwise, if the result of (12) is 1, (10) yields $m = R$ and the gate label masks $k_{\gamma,1} = k_{\gamma,0} \oplus R$. These are the expected results for an AND operation.
2) Let $\lambda_\gamma = 1$. If the result of (12) is 0, (10) yields $m = R$ and the gate label masks $k_{\gamma,1} = k_{\gamma,0} \oplus R$. Since $k_{\gamma,1}$ hides signal 0, this is as expected. Otherwise, if the result of (12) is 1, (10) yields $m = 0$ and the gate label masks $k_{\gamma,0}$. Again, the actual meaning of the signal $k_{\gamma,0}$ is inverted, i.e., 1 in this case, and the result is correct.

This shows that the AND gate correctly assigns the input signals to the corresponding output signals. Gate evaluation works by the same principle as we described in Section 2.2.2.

3.2.2. XOR Gates. XOR gates in the modified garbled circuits do not hold any labels. Let an XOR gate have input wires α and β, as well as output wire γ. An evaluator of the circuit simply has to calculate

$$k_{\gamma,(u\oplus\lambda_\alpha)\oplus(v\oplus\lambda_\beta)} = k_{\alpha,u\oplus\lambda_\alpha} \oplus k_{\beta,v\oplus\lambda_\beta} \tag{13}$$

on their input signals to obtain the output [5]. That this yields the correct output semantics for an XOR operation has been shown in (8). The following equations proof that the correct signals are calculated for all input cases [8].

$$\begin{aligned} k_{\gamma,0} &= k_{\alpha,0} \oplus k_{\beta,0} = (k_{\alpha,0} \oplus R) \oplus (k_{\beta,0} \oplus R) \\ &= k_{\alpha,1} \oplus k_{\beta,1} \\ k_{\gamma,1} &= k_{\gamma,0} \oplus R = k_{\alpha,0} \oplus (k_{\beta,0} \oplus R) = k_{\alpha,0} \oplus k_{\beta,1} \\ &= k_{\alpha,0} \oplus (k_{\beta,0} \oplus R) = (k_{\alpha,0} \oplus R) \oplus k_{\beta,0} \\ &= k_{\alpha,1} \oplus k_{\beta,0} \end{aligned}$$

They follow directly from the signal definitions.

4. Considerations for Using the BMR Protocol

In comparison to protocols like GMW and BGW, BMR's advantage of needing only a constant number of communication rounds for the circuit creation makes it a better fit for scenarios where communication between the parties is of comparatively more concern than local computation capabilities. For example, this is the case when SMC over the internet is required [5].

In scenarios with low network latency, protocols requiring a non-constant amount of communication rounds but less expensive processing, like the GMW protocol, could provide better overall performance than the BMR protocol [5].

It has to be noted that the original unmodified design of the BMR protocol is not suitable for real-world applications, because some details of it, like the usage of many zero-knowledge-proofs, are not efficiently computable, as has been noted by [7]. Since its introduction, however, much effort has successfully been spent to overcome those performance pitfalls. Eventually, reasonably efficient concrete real-world implementations of the protocol, like FairplayMP, introduced by A. Ben-David et al. in 2008 [6], have been developed.

5. Conclusion

We presented an expressive description of the basic BMR protocol for SMC, which enables the participation of any number of parties in a malicious setting. Its characteristic of requiring only a fixed constant number of rounds to create the garbled circuit makes it interesting for concrete real-world adaptations that use high latency communication over the internet. These implementations often introduce various optimizations to enhance its performance. The FreeXOR technique, for example, makes creation and evaluation expenses of XOR gates negligible in order to substantially boost performance.

References

[1] D. Evans, K. Vladimir, and M. Rosulek, "A pragmatic introduction to secure multi-party computation," *Foundations and Trends® in Privacy and Security*, vol. 2, no. 2-3, pp. 70–246, 2018. [Online]. Available: http://dx.doi.org/10.1561/3300000019

[2] A. C. Yao, "Protocols for secure computations," in *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*. IEEE, 1982, pp. 160–164. [Online]. Available: https://doi.org/10.1109/SFCS.1982.38

[3] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, ser. STOC '87. New York, NY, USA: Association for Computing Machinery, 1987, p. 218–229. [Online]. Available: https://doi.org/10.1145/28395.28420

[4] D. Beaver, S. Micali, and P. Rogaway, "The round complexity of secure protocols," in *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, ser. STOC '90. New York, NY, USA: Association for Computing Machinery, 1990, p. 503–513. [Online]. Available: https://doi.org/10.1145/100216.100287

[5] A. Ben-Efraim, Y. Lindell, and E. Omri, "Optimizing semi-honest secure multiparty computation for the internet," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 578–590. [Online]. Available: https://doi.org/10.1145/2976749.2978347

[6] A. Ben-David, N. Nisan, and B. Pinkas, "Fairplaymp: A system for secure multi-party computation," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, ser. CCS '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 257–266. [Online]. Available: https://doi.org/10.1145/1455770.1455804

[7] Y. Lindell, B. Pinkas, N. P. Smart, and A. Yanai, "Efficient constant-round multi-party computation combining bmr and spdz," *Journal of Cryptology*, vol. 32, no. 3, pp. 1026–1069, Jul 2019. [Online]. Available: https://doi.org/10.1007/s00145-019-09322-2

[8] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free xor gates and applications," in *Automata, Languages and Programming*, L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 486–498. [Online]. Available: https://doi.org/10.1007/978-3-540-70583-3_40