Rate-1 Arithmetic Garbling From Homomorphic Secret Sharing

Pierre Meyer[1], Claudio Orlandi[1], Lawrence Roy[1], and Peter Scholl[1]

Aarhus University, Aarhus, Denmark
{pierre.meyer,orlandi,peter.scholl}@cs.au.dk, ldr709@gmail.com

Abstract. We present a new approach to garbling arithmetic circuits using techniques from homomorphic secret sharing, obtaining constructions with high rate that support free addition gates. In particular, we build upon non-interactive protocols for computing distributed discrete logarithms in groups with an easy discrete-log subgroup, further demonstrating the versatility of tools from homomorphic secret sharing. Relying on distributed discrete log for the Damgård-Jurik cryptosystem (Roy and Singh, *Crypto '21*), whose security follows from the decisional composite residuosity assumption (DCR), we get the following main results:

– **Two ciphertexts per multiplication, from IND-CPA security of Damgård-Jurik.** Assuming the Damgård-Jurik cryptosystem is semantically secure (which follows from DCR), there is a garbling scheme for circuits with B-bounded integer arithmetic using only two ciphertexts per multiplication. The total bit-size of the resulting garbled circuit is:
$$(n + 2s_\times + 2D_\times) \cdot (\zeta + 1) \cdot \log N$$
where n is the number of inputs, $s_\times$ is the number of multiplications, $D_\times$ is the multiplicative depth of the circuit, N is an RSA modulus and $N^{\zeta-1}$ is a rough bound on the magnitude of wire values in the computation.

– **One ciphertext per multiplication, from KDM security of Damgård-Jurik.** Assuming the Damgård-Jurik encryption scheme remains secure given encryption of the key and its inverse, the construction achieves rate-1. The total bit-size of the resulting garbled circuit is:
$$(n + s_\times + 1) \cdot (\zeta + 1) \cdot \log N$$
where the parameters are as above, except $N^{\zeta-2}$ is the magnitude bound.

As a side result, we show that our scheme based on IND-CPA security achieves rate 3/5 for levelled circuits.

Table of Contents

1 Introduction

Garbled circuits are a tool commonly used in protocols for secure two-party computation and other cryptographic applications. Starting with the work of Yao [Yao82], most constructions of garbled circuits are in the setting of Boolean circuits: they transform a description of a circuit $C : \{0,1\}^n \to \{0,1\}$ into a garbled circuit $\widehat{C}$, together with some input encoding information $\boldsymbol{L} = \{L_{i,0}, L_{i,1}\}_{i=1}^n$. For an input x, an evaluator can use the garbled circuit together with the subset of encoded inputs $\boldsymbol{L}^x := \{L_{i,x_i}\}_i$ to learn $C(x)$, but nothing else about the circuit or the input (besides some structure of C). Yao's construction, and most subsequent works, incur a multiplicative overhead $O(\lambda)$ in the security parameter λ, meaning that the size of $\widehat{C}$ is $O(\lambda)$ times larger than the number of gates in the original circuit C.

To garble a function containing arithmetic operations such as integer addition and multiplication, traditionally one would have to first express these operations as Boolean circuits and then garble the resulting circuit. Known circuits for ℓ-bit integer multiplication have size $O(\ell \log \ell)$, and more commonly $O(\ell^2)$, which adds a large overhead to the size of the garbled circuit for these types of operations. This changed with the work of Applebaum, Ishai and Kushilevitz [AIK11], who gave the first direct methods for garbling arithmetic computations. They presented constructions in the model of arithmetic circuits for *bounded integer computation*, where there exists a (possibly exponential) bound $B \in \mathbb{N}$ such that for any set of admissible inputs, no wire value in the computation exceeds B in absolute value.

Their main construction builds upon information-theoretic techniques for garbling low-depth arithmetic circuits, and combines this with the learning with errors assumption to support arbitrary polynomial-size circuits. They give an alternative construction (with larger overhead) based solely on one-way functions, which combines classical Boolean garbled circuits with the Chinese remainder theorem.

Rate of Arithmetic Garbling. An important efficiency metric of a garbling scheme is the *rate*, which measures its bandwidth efficiency. In the model of B-bounded integer computation, where the bit-length of wire values is $\ell = \log(2B+1)$, for a circuit with n inputs and $|C|$ gates and garbled circuit of size $\mathsf{size}(\widehat{C})$ bits, the rate is roughly captured by the quantity

$$\mathsf{rate} = \frac{(|C|+n)\ell}{\mathsf{size}(\widehat{C}) + \mathsf{size}(\boldsymbol{L})}$$

We focus on measuring the rate as the circuit size tends towards infinity faster than the number of inputs, with worst-case circuits consisting of e.g. predominantly multiplication gates. We are also interested in the asymptotic behaviour of the rate as ℓ tends to ∞.

Using Yao's scheme, one can build an arithmetic garbling scheme with rate $O(1/(\lambda \log \ell))$ via asymptotically efficient (yet impractical) multiplication algorithms, or $O(1/(\lambda\ell))$ with schoolbook multiplication, relying only on the existence of one-way functions. The LWE-based construction of [AIK11] achieves constant rate for constant-degree polynomials, while for general circuits obtains rate $O(1/\lambda_{\mathsf{LWE}})$, where λ_{LWE} is the underlying LWE dimension. In very recent work, Heath [Hea24] gave the first construction of arithmetic garbling with rate $O(1/\lambda)$ that solely relies on "Minicrypt"-style assumptions, in this case a circular correlation-robust hash. Another recent work of Li and Liu [LL24] obtains rate $O(1/(\lambda\ell^{0.5}))$ in the random oracle model, as well as a construction with rate $O(\log \ell/(\lambda\ell))$ that supports free addition, matching the asymptotic efficiency of a construction from [BMR16], whilst additionally supporting improved bit decomposition gates.

The first *constant-rate* arithmetic garbling scheme for bounded integer computations was constructed by Ball, Li, Lin and Liu [BLLL23], under the decisional composite residuosity (DCR) assumption. Their construction uses the Damgård-Jurik [DJ01] extension of Paillier encryption [Pai99] to construct a more efficient key extension gadget than the LWE-based one of [AIK11], such that each garbled addition or multiplication gate requires only a constant number of Damgård-Jurik ciphertexts. When the integer bit-length is sufficiently large, Damgård-Jurik ciphertexts have rate $1-\varepsilon$, which leads to a constant-rate garbling scheme.

1.1 Related Work

Goldwasser et al. [GKP$^+$13] constructed reusable garbled circuits based on the subexponential learning with errors assumption, which was later improved to be *compact* by Boneh et al. [BGG$^+$14]. Compactness means that the size of the garbled circuit only scales polynomially with the depth of the circuit and not its size (assuming the cleartext circuit is known to the evaluator). This leads to a garbling scheme for Boolean or arithmetic circuits with rate *better than 1.* However, as observed in [BLLL23], the dependence on the circuit depth and security parameter are both very large (n^6) and the construction relies on heavy machinery including fully homomorphic encryption and attributed-based encryption, using subexponentially hard LWE.

The work of [BLLL23] additionally includes constructions of arithmetic garbling for modulo p computations, instead of bounded integers. They present constructions based on LWE and DCR, however, both with a low rate in $\tilde{O}(1/\lambda)$. Li and Liu [LL24] also presented constructions based on LWE and DCR, and in particular showed how to obtain free addition under DCR, albeit with worse communication for multiplication gates than [BLLL23]. Both of these works also present constructions that support bit-decomposition gadgets for mixing Boolean and arithmetic computations in the same circuit.

Using bilinear maps, [FMS19] gave a construction of arithmetic garbling for inner products with constant rate. Their construction can be bootstrapped to polynomial-sized circuits using the CRT-based compiler of [AIK11], however, this results in a poor rate.

Recently, Hazay and Yang [HY24] investigated the feasibility of maliciously secure garbling, building on the semi-honest construction of [BLLL23].

1.2 Our Contributions

We introduce a new approach to garbling arithmetic circuits using techniques from *homomorphic secret sharing* [BGI16, OSY21, RS21]. Unlike prior constructions [AIK11, BLLL23], which rely on the linearly homomorphic properties of Damgård-Jurik or LWE encryption, we additionally exploit methods for computing *distributed discrete logarithms* in groups with an easy discrete-log subgroup. Through this, we depart from the approach of combining an information-theoretic randomized encoding with a key-extension gadget [AIK11, BLLL23], instead giving direct constructions of arithmetic garbling with high rate.

We present two main constructions with security based on the Damgård-Jurik encryption scheme: (1) a construction with rate 1, based on the key-dependent message (KDM) security of Damgård-Jurik, and (2) a construction with rate 1/2, based on the decisional composite residuosity assumption (i.e. the IND-CPA security of Damgård-Jurik). The former result is the first rate-1 construction of arithmetic garbled circuits that does not rely on heavy tools such as attribute-based encryption. Both constructions also enjoy free addition gates.

The asymptotic efficiency of our constructions is compared with prior works in Table 1. We also give concrete examples for $\ell \approx \log N$ bits, where N is the RSA modulus, The concrete values for the rate of the [BLLL23] constructions come from [BLLL23, Table 3], which uses an integer bound of $B \approx 4000$ bits and a 4096-bit Paillier modulus N. Notice that our two constructions achieve concrete rates of 1/4 and 1/6 using the same parameters. As ℓ grows larger, the rate of our schemes quickly approaches 1 and 1/2, for instance when $\ell \approx 12000$ and $\log N \approx 2000$, we obtain respective rates 1/2 and 3/10. Meanwhile, the constructions from [BLLL23] have rate 1/12 at best.

Construction	Security	Rate	Free addition
[AIK11]	LWE	$\Theta(1/\lambda_{\mathsf{LWE}})$	✗
Yao+ [HVDH21]	OWF	$\Theta(1/\lambda \log \ell)$	✗
[BMR16, LL24]	RO	$\Theta(\log \ell / \lambda \ell)$	✓
[Hea24]	CCR hash	$\Theta(1/\lambda)$	✗
[BLLL23]	DCR	$\Theta(1/c)$	✗
(e.g. $\ell \approx \log N$)		1/36	
[BLLL23]	strong DCR	$\Theta(1/c)$	✗
(e.g. $\ell \approx \log N$)		1/24	
Section 4.1	DCR + KDM	$1 - \varepsilon$	✓
(e.g. $\ell \approx \log N$)		1/4	
Section 4.2	DCR	$1/2 - \varepsilon$	✓
(e.g. $\ell \approx \log N$)		1/6	

Table 1: Summary of bounded integer arithmetic garbling schemes for ℓ-bit integers. N is an RSA modulus, $c > 1$ is a constant, ε is a constant approaching zero as $\ell \to \infty$. "Strong DCR" is a variant of DCR combined with a DDH-like assumption with small exponent

2 Technical Overview

Given $x \in \mathbb{Z}$, we call "subtractive shares of x", denoted $\langle x\rangle_1$ and $\langle x\rangle_0$, values which satisfy $\langle x\rangle_1 - \langle x\rangle_0 = x$. We further use $\langle x\rangle$ to mean "either $\langle x\rangle_1$ or $\langle x\rangle_0$, as clear from context". In particular, if the evaluator and the garbler hold $\langle x\rangle_1$ and $\langle x\rangle_0$ respectively, then having parties compute $f(\langle x\rangle)$ means having them compute $f(\langle x\rangle_1)$ and $f(\langle x\rangle_0)$ respectively.

In Section 2.1, we present a simple, warm-up construction that shows how to build high-rate arithmetic garbled circuits using efficient distributed discrete logarithm (DDLog) algorithms. In Section 2.2, we describe an optimised scheme, which achieves rate-$(1-\varepsilon)$ if the DDLog's underlying encryption scheme is secure against *key-dependent messages* (KDM-secure). In Section 2.3, we instantiate this optimised construction using the Damgård-Jurik cryptosystem (and associated DDLog). In Section 2.4, we provide a rate-$\frac{1-\varepsilon}{2}$ garbling scheme, relying only on semantic security (and not KDM-security) of Damgård-Jurik.

2.1 Arithmetic garbled circuits through the lense of homomorphic secret sharing

We build arithmetic garbled circuits following the definitional template introduced by Applebaum, Ishai, and Kushilevitz [AIK11]. The garbler (who knows the circuit but not its inputs) generates a global key $\boldsymbol{k}$ and a wire-dependent key $\boldsymbol{K}_x$ for each wire x. Each wire x is then associated with the label $\boldsymbol{L}_x = \boldsymbol{k}\cdot x + \boldsymbol{K}_x$. Note that this can also be interpreted as a secret-sharing $\langle \boldsymbol{k}\cdot x\rangle$ where $\langle \boldsymbol{k}\cdot x\rangle_1 = \boldsymbol{L}_x$ is known to the evaluator and $\langle \boldsymbol{k}\cdot x\rangle_0 = \boldsymbol{K}_x$ is known to the garbler. We will exploit this duality to use techniques originating in the *homomorphic secret-sharing (HSS)* domain to construct our garbling schemes.

Jumping ahead, the evaluator must be able to retrieve the values corresponding to the labels of the output wires. To allow this, we let the garbler sample $\boldsymbol{k}$ as a vector whose first coordinate is 1: it now follows that for each wire x, $\boldsymbol{L}_x.\mathsf{fst} = x + \boldsymbol{K}_x.\mathsf{fst}$. For each output value z, including $\boldsymbol{K}_z.\mathsf{fst}$ in the garbled circuit therefore allows the evaluator to retrieve z from $\boldsymbol{L}_z$. As it turns out, defining $\boldsymbol{k}$ like this also helps us with the gate-by-gate evaluation.

The wire-dependent key $\boldsymbol{K}_x$ is instead generated uniformly at random for each input x, and then the garbler proceeds to define the other keys in a gate-by-gate fashion. Considering the "HSS viewpoint", our task can then be viewed as having garbler and evaluator non-interactively convert subtractive shares $\langle \boldsymbol{k}\cdot x\rangle$ and $\langle \boldsymbol{k}\cdot y\rangle$ into subtractive shares $\langle \boldsymbol{k}\cdot g(x,y)\rangle$ for any given gate $g \in \{+,-,\times\}$.

If g is an addition gate with output $z = x+y$, the garbler can simply define their share $\boldsymbol{K}_z$ as $\boldsymbol{K}_z \coloneqq \boldsymbol{K}_x + \boldsymbol{K}_y$. This allows the evaluator to compute their share $\boldsymbol{L}_z = \boldsymbol{L}_x + \boldsymbol{L}_y$, which equals $\boldsymbol{k}\cdot(x+y) + (\boldsymbol{K}_x + \boldsymbol{K}_y)$. From the HSS viewpoint, this works since $\langle x+y\rangle = \langle x\rangle + \langle y\rangle$.

If g is a multiplication gate with output $z = x\cdot y$, we will exploit the following identity:

$$\begin{aligned}\boldsymbol{k}\cdot z = \boldsymbol{k}x\cdot y &= (\boldsymbol{k}x + \boldsymbol{K}_x)\cdot(y + \boldsymbol{K}_y.\mathsf{fst}) - \boldsymbol{k}x\cdot(\boldsymbol{K}_y.\mathsf{fst}) - y\cdot\boldsymbol{K}_x - \boldsymbol{K}_x\cdot(\boldsymbol{K}_y.\mathsf{fst})\\ &= \boldsymbol{L}_x\cdot(\boldsymbol{L}_y.\mathsf{fst}) - \boldsymbol{k}x\cdot(\boldsymbol{K}_y.\mathsf{fst}) - y\cdot\boldsymbol{K}_x - \boldsymbol{K}_x\cdot(\boldsymbol{K}_y.\mathsf{fst}) \qquad (1)\end{aligned}$$

The terms $\boldsymbol{L}_x\cdot(\boldsymbol{L}_y.\mathsf{fst})$ and $\boldsymbol{K}_x\cdot(\boldsymbol{K}_y.\mathsf{fst})$ can be computed locally by the evaluator and garbler respectively, but neither can compute $\boldsymbol{k}x\cdot(\boldsymbol{K}_y.\mathsf{fst})$ or $y\cdot\boldsymbol{K}_x$. We therefore let the garbler provide appropriate encodings of $(\boldsymbol{K}_y.\mathsf{fst})$ and $\boldsymbol{K}_x$ as part of the garbled circuit, so that the evaluator can compute shares of these terms, by solving the *distributed discrete logarithm problem (DDLog)*.

Share conversion from distributed discrete logarithm. Without going into details, for now we assume the existence of a deterministic algorithm DDLog, that both the garbler and the evaluator can locally execute, satisfying the following property (for some appropriately chosen encryption scheme):

> Let $c = \mathsf{Enc}_{\mathsf{pk}}(A)$ be an encryption of A with decryption key sk, and $\langle \mathsf{sk} \cdot B \rangle$ a sharing (over $\mathbb{Z}$) of $\mathsf{sk} \cdot B$, then $\mathsf{DDLog}(c^{\langle \mathsf{sk} \cdot B \rangle}) = \langle \mathsf{sk} \cdot AB \rangle$ is a subtractive share (over $\mathbb{Z}$) of $\mathsf{sk} \cdot AB$.
> In particular, if c is an encryption of sk^{-1} (with the inverse taken with respect to the plaintext modulus) then it holds that $\mathsf{DDLog}(c^{\langle \mathsf{sk} \cdot B \rangle}) = \langle B \rangle$.

(The reader already familiar with homomorphic secret sharing literature may pause here, as it is more usual for $\mathsf{DDLog}(\mathsf{Enc}_{\mathsf{pk}}(A)^{\langle \mathsf{sk} \cdot B \rangle})$ to be a share of AB, not $\mathsf{sk} \cdot AB$.)

Garbling a multiplication gate. We first specify how $\boldsymbol{k}$ is defined: the garbler samples a keypair $(\mathsf{sk}, \mathsf{pk})$ associated with a DDLog-compatible encryption scheme, then sets $\boldsymbol{k} \leftarrow (1, \mathsf{sk})$. To garble a multiplication gate with input wire keys $\boldsymbol{K}_x, \boldsymbol{K}_y$, the garbler computes the following ciphertexts and adds them to the garbled circuit[1]:

$$c \xleftarrow{\$} \mathsf{Enc}_{\mathsf{pk}}(\mathsf{sk}^{-1}),\ c_y \xleftarrow{\$} \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{K}_y.\mathsf{fst}),\ \text{and } \boldsymbol{c}_x \xleftarrow{\$} \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{K}_x)\ .$$

Using this, the garbler and evaluator can do the following:

1. With their shares $\langle \mathsf{sk} \cdot x \rangle$ (namely, $\boldsymbol{K}_x.\mathsf{snd}$ and $\boldsymbol{L}_x.\mathsf{snd}$) and ciphertext c_y encrypting $\boldsymbol{K}_y.\mathsf{fst}$, run DDLog to obtain shares $\langle \mathsf{sk} \cdot x \cdot \boldsymbol{K}_y.\mathsf{fst} \rangle$.
2. Using shares $\langle \mathsf{sk} \cdot x \cdot \boldsymbol{K}_y.\mathsf{fst} \rangle$ and c encrypting sk^{-1}, run DDLog to get shares $\langle x \cdot \boldsymbol{K}_y.\mathsf{fst} \rangle$.
3. Concatenate the resulting shares $\langle x \cdot \boldsymbol{K}_y.\mathsf{fst} \rangle$ and $\langle \mathsf{sk} \cdot x \cdot \boldsymbol{K}_y.\mathsf{fst} \rangle$, to get $\langle \boldsymbol{k} x \cdot \boldsymbol{K}_y.\mathsf{fst} \rangle$.

Similarly, they can:

4. Use their shares $\langle \mathsf{sk} \cdot y \rangle$ (namely, $\boldsymbol{K}_y.\mathsf{snd}$ and $\boldsymbol{L}_y.\mathsf{snd}$) and the encryption $\boldsymbol{c}_x$ of $\boldsymbol{K}_x$, and run DDLog to compute shares $\langle \mathsf{sk} \cdot y \cdot \boldsymbol{K}_x \rangle$.
5. Use shares $\langle \mathsf{sk} \cdot y \cdot \boldsymbol{K}_x \rangle$ and the encryption of sk^{-1}, the parties compute shares $\langle y \cdot \boldsymbol{K}_x \rangle$.

If the garbler and the evaluator define $\boldsymbol{K}_z$ and $\boldsymbol{L}_z$ respectively by offsetting $\boldsymbol{K}_x \cdot (\boldsymbol{K}_y.\mathsf{fst})$ and $\boldsymbol{L}_x \cdot (\boldsymbol{L}_y.\mathsf{fst})$ by their shares of $\boldsymbol{k} x \cdot \boldsymbol{K}_y.\mathsf{fst}$ (step 3) and $y \cdot \boldsymbol{K}_x$ (step 5), then, by Eq. (1), the invariant "$\boldsymbol{L}_z = \boldsymbol{k} \cdot z + \boldsymbol{K}_z$" is maintained.

Summarising, the garbled circuit contains one globally reusable ciphertext (c, the encryption of sk^{-1}), one ciphertext $\boldsymbol{c}_x$ per left input-wire to a multiplication, one ciphertext c_y per right input-wire to a multiplication, and one mask $\boldsymbol{K}_z.\mathsf{fst}$ per output. Overall, provided the encryption scheme has rate-1, the garbling scheme has rate close to 1/2 (provided the secret key is small enough compared to the ring over which the circuit's arithmetic is performed[2]) and achieves free addition.

2.2 Achieving rate-1

We now show how the above (sketch of a) construction can be adapted to achieve rate-1, while preserving free addition. There are two key ideas. The first is an alternative identity to Eq. (1) which breaks the asymmetry between the left and right wires of multiplication gates, which means all ciphertexts are of the same form. The second is exploiting linear homomorphism

[1] Note that the first two ciphertexts encrypt scalars, while the last encrypts a tuple.
[2] We need *authenticated* shares of wire values (*i.e.* $\langle \mathsf{sk} \cdot w \rangle$) to fit in the plaintext space.

of the underlying encryption scheme in order to halve the number of ciphertexts: In the previous construction, the garbled circuit includes one ciphertext for each input wire to a multiplication gate. In the following, the garbled circuit only contains one ciphertext for each multiplication gate (which this time depends on the output wire of the gate), as well as one ciphertext per input wire.

In the rate-1 construction, $k \leftarrow \mathsf{sk}$, L_x and K_x (for each wire x) are now scalars satisfying $k \cdot x = L_x - K_x$. We handle addition gates exactly as before. To understand how we support multiplications, consider Eq. (2):

$$\begin{aligned}\mathsf{sk}^2 \cdot z = \mathsf{sk} \cdot x \cdot \mathsf{sk} \cdot y &= (\mathsf{sk} \cdot x + K_x) \cdot (\mathsf{sk} \cdot y + K_y) - \mathsf{sk} \cdot x \cdot K_y - \mathsf{sk} \cdot y \cdot K_x - K_x \cdot K_y \\ &= L_x \cdot L_y - \mathsf{sk} \cdot x \cdot K_y - \mathsf{sk} \cdot y \cdot K_x - K_x \cdot K_y \qquad (2)\end{aligned}$$

Including encryptions of K_x and K_y in the garbled circuit allows the garbler and evaluator to use DDLog to compute shares $\langle \mathsf{sk} \cdot x \cdot K_y \rangle$ and $\langle \mathsf{sk} \cdot y \cdot K_x \rangle$ since they hold shares $\langle \mathsf{sk} \cdot x \rangle$ (namely, L_x and K_x) and $\langle \mathsf{sk} \cdot y \rangle$ (namely, L_y and K_y). In turn, using Eq. (2), the garbler and the evaluator can compute shares $\langle \mathsf{sk}^2 \cdot z \rangle$ (recall that the garbler can compute $K_x \cdot K_y$ and the evaluator can compute $L_x \cdot L_y$). Finally, including an encryption of sk^{-1} in the garbled circuit allows the parties to use their shares $\langle \mathsf{sk}^2 \cdot z \rangle$ and DDLog to compute shares $\langle \mathsf{sk} \cdot z \rangle$: these shares define K_z and L_z. Furthermore, if z is an output gate, the parties can use DDLog once more, again with the encryption of sk^{-1}, to compute shares $\langle z \rangle$; as before, if the garbler publishes its share of z then the evaluator can reconstruct the value of z from the label L_z.

One question remains: how do we obtain the encryptions of K_x and K_y for each multiplication gate $z = x \times y$, without publishing two ciphertexts per gate? By definition, $K_{x+y} := K_x + K_y$, and therefore if the encryption scheme supports linear homomorphism, the parties can use encryptions of K_x and K_y to generate an encryption of K_{x+y}. So, it is enough for the garbler to send an encryption of K_w for each w which is either an input to the circuit or the output of a multiplication gate: the parties can then proceed in a gate-by-gate fashion to reconstruct all the necessary ciphertexts.

2.3 Concrete instantiation using the Damgård-Jurik encryption scheme [DJ01]

Recall that, in the Damgård-Jurik cryptosystem [DJ01], the public key is an RSA modulus $N = p \cdot q$ (where p and q are large primes) while the secret key is its Euler totient $\mathsf{sk} = (p-1) \cdot (q-1)$. The scheme is parameterised by an integer $\zeta \geq 2$: the plaintext space is $\mathbb{Z}/N^{\zeta}\mathbb{Z}$, while the ciphertext space is $\mathbb{Z}/N^{\zeta+1}\mathbb{Z}$. The encryption scheme therefore has rate $1/(1 + 1/\zeta)$. The scheme is also linearly homomorphic, and Roy and Singh [RS21] showed there exists a DDLog algorithm satifying the requirements from Section 2.1)[3], so it can be used to instantiate the arithmetic garbled circuit described in Section 2.2. Because we need to fit (shares of) $\mathsf{sk} \cdot w$ in the plaintext space (of size N^{ζ}) for each wire w (and $\mathsf{sk} = (p-1) \cdot (q-1) \leq p \cdot q = N$), we can set parameters so the arithmetic is performed over a ring of size $N^{\zeta-1}$. However, we cannot use this entire plaintext space, due at a failure probability in DDLog when the underlying plaintexts are too large. This means we get a garbling scheme for B-bounded integer computation, where $B \approx N^{\zeta-1}/2^{\kappa}$ and κ is a statistical correctness parameter (ensuring correctness with probability $1 - 2^{-\kappa}$ per gate). Overall, the garbling scheme of Section 2.2 has rate $(\zeta - 1)/(\zeta + 1)$. We can therefore achieve rate-$(1 - \varepsilon)$ for an arbitrarily small constant ε.

Figure 1 summarises how the garbler and the evaluator perform multiplications in the rate-1 garbling scheme of Section 2.2 as instantiated with Damgård-Jurik (which we assume

[3] That is, if c is an encryption of y under public key N, and $\langle \mathsf{sk} \cdot x \rangle$ is a share of $\mathsf{sk} \cdot x$, then $\mathsf{DDLog}(c^{\langle \mathsf{sk} \cdot x \rangle})$ is a share of $\mathsf{sk} \cdot xy$.

to be KDM-secure) and Roy-Singh's DDLog. For clarity, the figure shows the roles of the two parties as symmetric, but recall that we are instantiating a garbling scheme, where the garbler can perform all of their garbling operations first, send the garbled circuit to the evaluator, who then evaluates the gates.

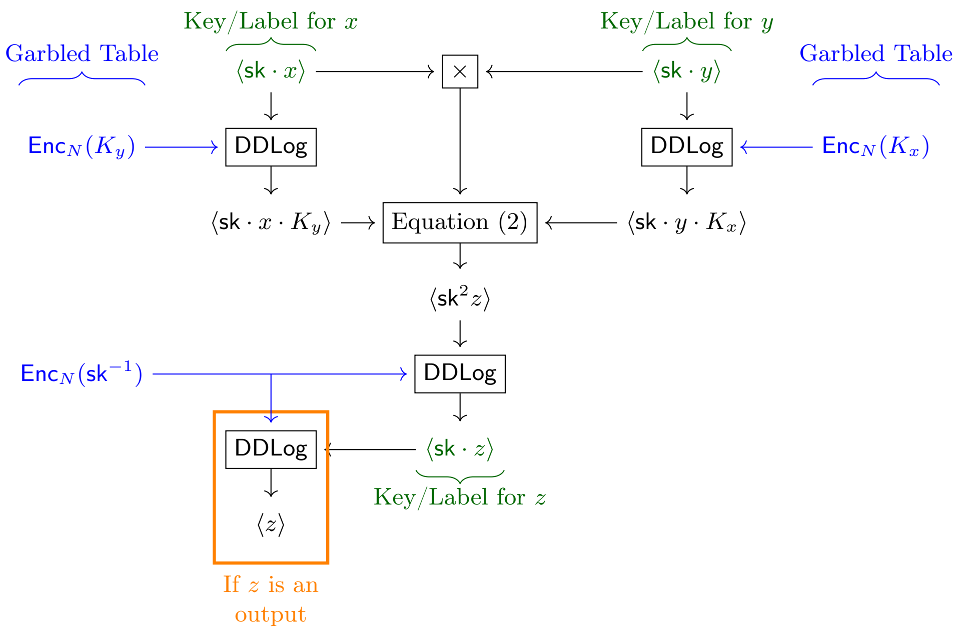

Key/Label for x
Key/Label for y
Garbled Table
Garbled Table
$\langle \mathsf{sk} \cdot x \rangle$
$\times$
$\langle \mathsf{sk} \cdot y \rangle$
$\mathsf{Enc}_N(K_y)$
DDLog
DDLog
$\mathsf{Enc}_N(K_x)$
$\langle \mathsf{sk} \cdot x \cdot K_y \rangle$
Equation (2)
$\langle \mathsf{sk} \cdot y \cdot K_x \rangle$
$\langle \mathsf{sk}^2 z \rangle$
$\mathsf{Enc}_N(\mathsf{sk}^{-1})$
DDLog
DDLog
$\langle \mathsf{sk} \cdot z \rangle$
Key/Label for z
$\langle z \rangle$
If z is an output

Fig. 1: Operations performed by the garbler (resp. evaluator) to convert their keys (resp. labels) for wires x and y into a key (resp. label) for wires $z = x \cdot y$. This requires KDM-security of Damgård-Jurik encryption; (sk, N) is the secret/public key pair.

2.4 Removing the circular-security assumption

The security of the scheme in Section 2.3 relies on assuming security of the underlying encryption scheme against *key-dependent message* attacks (KDM-security). Most visibly, this is because the garbled circuit contains an encryption of the inverse of the secret key, but also because the evaluator receives encryptions of shares of the form $\mathsf{Enc}_{\mathsf{pk}}(K_x = \langle \mathsf{sk} \cdot x \rangle_0)$ as well as receiving their share $L_x = \langle \mathsf{sk} \cdot x \rangle_1$ of this value. The evaluator may know x, and x would allow computing $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{sk})$ from $\mathsf{Enc}_{\mathsf{pk}}(K_x)$ because $(L_x - K_x)/x = \mathsf{sk}$ and Enc is additively homomorphic. Our scheme can be adapted to instead rely on semantic security (provably implied by the *decisional composite residuosity* assumption, DCR), but at the cost of reducing the rate to $1/2 - \varepsilon$.

Removing the need for $\mathsf{Enc}_N(\mathsf{sk}^{-1})$**.** Without circular security, we can no longer use an encryption of sk^{-1} to convert authenticated wire shares $\langle \mathsf{sk} \cdot w \rangle$ into plain shares $\langle w \rangle$ (which are required for output wires). Instead, following [RS21], we will use an alternative decryption key for Damgård-Jurik, which allows directly obtain shares $\langle w \rangle$ from DDLog without an encryption of sk^{-1}:

Let $d \leftarrow \mathsf{sk} \cdot (\mathsf{sk}^{-1} \bmod N^{\zeta})$ and c be a Damgård-Jurik encryption of y under public key N, then $\mathsf{DDLog}(c^{\langle d \cdot x \rangle}) = \langle x \cdot y \rangle$ is a share of $x \cdot y$.

Next, to use this in garbling, we will use a third multiplication identity:[4]

$$z = x \cdot y = \langle x \rangle_1 \cdot \langle y \rangle_1 - x \cdot \langle y \rangle_0 - y \cdot \langle x \rangle_0 - \langle x \rangle_0 \cdot \langle y \rangle_0 \tag{3}$$

[4] Recall, we use subtractive share notation, where $\langle x \rangle$ means that $x = \langle x \rangle_1 - \langle x \rangle_0$.

Assuming the parties' shares can be derived from their key or label for x, the parties can locally compute the respective values $\langle x\rangle_0 \cdot \langle y\rangle_0$ and $\langle x\rangle_1 \cdot \langle y\rangle_1$. To obtain the missing cross-terms, the garbler can provide $\mathsf{Enc}_N(\langle w\rangle_0)$ for each wire w (or rather, exploiting linear homomorphism as in Section 2.2, one ciphertext for each input or output of a multiplication gate is enough), allowing use of DDLog to obtain shares of the necessary products.

The problem, however, is that we still need to get shares of $d \cdot w$ for each wire w. Trying to generate these shares in the same way as before runs again into a circularity issue, as an identity of the form

$$d \cdot z = \langle d \cdot x\rangle_1 \cdot \langle y\rangle_1 - dx \cdot \langle y\rangle_0 - y \cdot \langle d \cdot x\rangle_0 - \langle d \cdot x\rangle_0 \cdot \langle y\rangle_0$$

requires the garbler to provide an encryption of $\langle d \cdot x\rangle_0$. To bypass this problem, we instead generate a sequence of keypairs, one per multiplicative level of the circuit, and exploit the idea of *key-switching.*

Key-switching. Let D be the multiplicative depth of the circuit. For each $i \in [0, D]$, we have the garbler generate a keypair (N_i, sk_i) (and set d_i accordingly). The idea behind key-switching is to allow the parties to convert shares of $d_i \cdot w$ into shares of $d_{i+1} \cdot w$ (and recursively into shares of any shares of $d_j \cdot w$ for $j > i$). This can simply be done by having the garbler provide an encryption of d_{i+1} under keypair (N_i, sk_i). Then, the parties can compute shares:

$$\langle d_{i+1} \cdot w\rangle \leftarrow \mathsf{DDLog}(c_i^{\langle d_i \cdot w\rangle}), \text{ where } c \xleftarrow{\$} \mathsf{Enc}_{N_i}(d_{i+1}) \; .$$

Thanks to this trick, it is now possible to build an arithmetic garbling scheme without resorting to KDM-style assumptions, at the price of achieving a lower rate $\frac{1}{2} - \varepsilon$. Getting to the final construction requires taking care of more technical issues, which we explain in detail in Section 4.2.

3 Preliminaries

3.1 Arithmetic Garbled Circuits

3.1.1 Arithmetic circuits with bounded integer computation. We use a model of arithmetic circuits consisting of addition, subtraction and (fan-in two) multiplication gates, all computed over $\mathbb{Z}$. We work in the *bounded integer computation* model, where there exists a bound $B = B(\lambda) \in \mathbb{N}$ that bounds the magnitude of circuit wire values. For a circuit C with n input wires, we say that an input vector $\boldsymbol{x} \in \mathbb{Z}^n$ is *admissible with respect to bound B* if the inputs, outputs and every intermediate wire value obtained during the evaluation of C are in the range $[-B, B]$.

3.1.2 Arithmetic Garbling. We define arithmetic garbled circuits, adapting the definition of [BLLL23].

Definition 1 ((B-bounded) Arithmetic Garbled Circuit). *A garbling scheme for a family of circuits classes $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}^\star}$ for B-bounded integer computation with label space $\mathcal{L} = \mathcal{L}(\lambda)$ is a pair of* p.p.t. *algorithms* $\mathsf{AGC} = (\mathsf{AGC.Garble}, \mathsf{AGC.Eval})$ *with the following syntax and properties:*

- $\mathsf{Garble}(1^\lambda, C)$: *On input a security parameter 1^λ and a circuit $C \in \mathcal{C}_\lambda$ with n inputs,* Garble *outputs n key pairs $(\boldsymbol{k}_0^i, \boldsymbol{k}_1^i)_{i \in [n]} \in \mathcal{L}$ and a garbled circuit $\widehat{C}$.*
- $\mathsf{Eval}((\boldsymbol{L}_i)_{i \in [n]}, \widehat{C})$: *On input n input labels $(\boldsymbol{L}_i)_{i \in [n]} \in \mathcal{L}^n$, and a garbled circuit $\widehat{C}$,* Eval *outputs a value $y \in [0, B]$.*

– ***Correctness.*** *A garbling scheme is* correct *if there exists a negligible function* negl *such that for all $\lambda \in \mathbb{N}$, every circuit $C \in \mathcal{C}_\lambda$ with n inputs, and every $x_1, \dots, x_n \in \mathcal{X}^n$ where $\mathcal{X}$ is the set of admissible inputs of C induced by bound B, the following holds:*

$$\Pr\left[\mathsf{Eval}((\boldsymbol{L}_i)_{i\in[n]}, \widehat{C}) =_{\mathbb{Z}} C(x_1, \dots, x_n) \colon \begin{array}{r} ((\boldsymbol{k}_0^i, \boldsymbol{k}_1^i)_{i\in[n]}, \widehat{C}) \xleftarrow{\$} \mathsf{Garble}(1^\lambda, C) \\ \boldsymbol{L}_i \leftarrow \boldsymbol{k}_0^i \cdot x_i + \boldsymbol{k}_1^i \end{array}\right] \le \mathsf{negl}(\lambda) \ .$$

– ***Privacy.*** *A garbling scheme is* secure *if there exist a* p.p.t. *simulator $\mathcal{S}$ such that for all sequence of circuits $\{C_\lambda\}_{\lambda\in\mathbb{N}}$, where $C_\lambda \in \mathcal{C}_\lambda$ with $n = n(\lambda)$ inputs, and every admissible (with respect to B and C_λ) sequence of inputs $\{(x_{1,\lambda}, \dots, x_{n,\lambda})\}_{\lambda\in\mathbb{N}}$, the following holds:*

$$\left\{\mathcal{S}(1^\lambda, C_\lambda, y) \colon y \leftarrow C_\lambda(x)\right\} \overset{c}{\approx} \left\{((\boldsymbol{L}_i)_{i\in[n]}, \widehat{C}_\lambda) \colon \begin{array}{r} ((\boldsymbol{k}_0^i, \boldsymbol{k}_1^i)_{i\in[n]}, \widehat{C}_\lambda) \xleftarrow{\$} \mathsf{Garble}(1^\lambda, C_\lambda) \\ \boldsymbol{L}_i \leftarrow \boldsymbol{k}_0^i \cdot x_i + \boldsymbol{k}_1^i \\ y \leftarrow C_\lambda(x) \end{array}\right\} .$$

To measure the efficiency of an arithmetic garbled circuit, we define its *rate* as follows.

Definition 2 (Rate of Arithmetic Garbled Circuit). *Let $\mathcal{C}$ be a class of arithmetic circuits, let* AGC *be an arithmetic garbling scheme for $\mathcal{C}$ with B-bounded computation, and let $\ell = \log(2B + 1)$. The* rate *of* AGC *for $\mathcal{C}$ is the quantity:*

$$\mathsf{rate} = \liminf_{C \in \mathcal{C}} \min_{x} \frac{(|C| + n)\ell}{\mathsf{size}(\widehat{C}) + \sum_i \mathsf{size}(\boldsymbol{k}_0^i \cdot x_i + \boldsymbol{k}_1^i)},$$

where the minimum is taken over all admissible inputs to the circuit $\mathcal{C}$, and the limit infimum is taken over all circuits in $\mathcal{C}$ (in the sense of the limit of a net, as we require $\mathcal{C}$ to form a directed set when ordered by inclusion).

We are most interested in the case where the circuit is dominated by gates.[5] That is, we will typically be interested in classes of circuits $\mathcal{C}_f$ that contain only circuits where $n \le f(|C|)$ for a sublinear function f. Note that the rate is a worst-case quantity that does not account for optimizations like free addition gates.

Previously, [BLLL23] define rate to be roughly the inverse of the above quantity, meaning that typically rate ≥ 1. Our definition is more consistent with the standard concept of the rate of an encryption scheme, which is at most 1.

When AGC supports arbitrarily large values of ℓ, we are also interested in the asymptotic behaviour of the rate as ℓ tends to ∞.

3.2 Damgård-Jurik Cryptosystem

The Damgård-Jurik cryptosystem [DJ01] is a generalisation of the Paillier cryptosystem [Pai99] (which, with the notation of Fig. 2, can be seen as the special case $\zeta = 1$).

Definition 3 (Decision Composite Residuosity Assumption (DCR), [Pai99]). *Let* RSA.Gen *be a polynomial-time algorithm which, on input a security parameter λ, outputs (N, p, q) where p and q are λ-bit primes and $N = pq$. Let λ be a security parameter. We say that the* Decision Composite Residuosity (DCR) *problem is hard relative to modulus-sampling algorithm* RSA.Gen *if*

$$\left\{(N, x) \colon \begin{array}{r} (N, p, q) \xleftarrow{\$} \mathsf{RSA.Gen} \\ x \xleftarrow{\$} (\mathbb{Z}/N^2\mathbb{Z})^\times \end{array}\right\} \overset{c}{\approx} \left\{(N, x^N \bmod N) \colon \begin{array}{r} (N, p, q) \xleftarrow{\$} \mathsf{RSA.Gen} \\ x \xleftarrow{\$} (\mathbb{Z}/N^2\mathbb{Z})^\times \end{array}\right\} .$$

[5] As an alternative to restricting to circuits with more gates than inputs, we could allow the garbling scheme to use "compressed inputs". That is, if $\boldsymbol{k}_0^i$ is the same for all i, and $\boldsymbol{k}_1^i$ is sampled randomly by the functionality (so that it can be replaced by something pseudorandom), then don't charge for the size of the input wire labels. There are methods of generating the input wire labels using only sublinear communication, such as the succinct VOLE of [ARS24]. This modified rate definition would reflect that.

Theorem 4 (Damgård-Jurik Cryptosystem [DJ01]). *For any choice of the parameter $\zeta \geq 2$, the construction of Fig. 2 is a CPA-secure linearly homomorphic encryption scheme if and only if the DCR assumption holds.*

LHE Damgård-Jurik Cryptosystem [DJ01]

Requires:

- $\mathsf{RSA.Gen}(\cdot)$ is an RSA modulus generation algorithm which, on input a security parameter 1^λ, samples $(p, q) \xleftarrow{\$} [2^{\ell(\lambda)-1}, 2^{\ell(\lambda)}]$ (where ℓ is some polynomial chosen appropriately in order for the cryptosystem to achieve λ bits of security) then computes $N \leftarrow p \cdot q$, and outputs (N, p, q).
- $\zeta \geq 2$ is a constant defining the plaintext size.
- Functions $\exp\colon (\mathbb{Z}/N^\zeta\mathbb{Z})^+ \to 1 + N(\mathbb{Z}/N^{\zeta+1}\mathbb{Z})$ and $\log\colon 1 + N(\mathbb{Z}/N^{\zeta+1}\mathbb{Z}) \to (\mathbb{Z}/N^\zeta\mathbb{Z})^+$ are defined by the following expressions, as in [RS21]:

$$\exp(x) = \sum_{k=0}^{\zeta} \frac{(Nx)^k}{k!} \qquad \text{and} \qquad \log(1 + Nx) = \sum_{k=1}^{\zeta} \frac{(-N)^{k-1}x^k}{k}$$

$\mathsf{DJ.KeyGen}(1^\lambda)$:

1. Run $(N, p, q) \xleftarrow{\$} \mathsf{RSA.Gen}(1^\lambda)$
2. Compute $\varphi \leftarrow (p-1) \cdot (q-1)$
3. Output (N, φ)

$\mathsf{DJ.Enc}_{N,\zeta}(x)$:

1. Sample $r \xleftarrow{\$} (\mathbb{Z}/N^{\zeta+1}\mathbb{Z})^\times$
2. Compute $c \leftarrow r^{N^\zeta} \cdot \exp(x)$
3. Output c

$\mathsf{DJ.Eval}_{N,\zeta}(c_1, c_2, \alpha)$:

// Homomorph. eval. $(x, y) \mapsto \alpha \cdot x + y$

Compute and output $c \leftarrow c_1^\alpha \cdot c_2$

$\mathsf{DJ.Dec}_{N,\zeta,\varphi}(c)$:

1. Compute ψ as the multiplicative inverse of φ in $\mathbb{Z}/N^\zeta\mathbb{Z}$
2. Compute and output $x \leftarrow \psi \cdot \log(c^\varphi)$

Fig. 2: The Damgård-Jurik cryptosystem.

3.3 IND-KDM Security

Informally, an encryption scheme is KDM-secure if it remains secure when the plaintext messages are a function of the secret key.

Definition 5 (Key-dependent message security [BRS03], Special Case). *A public-key encryption scheme* $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*, whose private-key and message spaces are denoted $\mathcal{K}$ and $\mathcal{M}$ respectively, is* secure in the presence of key-dependent messages with respect to function class $\mathcal{F} \subseteq \mathcal{K} \to \mathcal{M}$ *(or simply* $\mathcal{F}$-KDM-secure*) if for every* p.p.t. *algorithm $\mathcal{A}^{\cdot}$ with oracle queries*

$$\left|\Pr\left[\mathcal{A}^{\mathcal{O}^{\mathsf{KDM}}_{\mathcal{F},\mathsf{sk},0}}(1^\lambda, \mathsf{pk}) = 1 \colon (\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)\right] - \Pr\left[\mathcal{A}^{\mathcal{O}^{\mathsf{KDM}}_{\mathcal{F},\mathsf{sk},1}}(1^\lambda, \mathsf{pk}) = 1 \colon (\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)\right]\right| \leq \mathsf{negl}(\lambda) \ ,$$

where oracles $\mathcal{O}^{\mathsf{KDM}}_{\mathcal{F},\mathsf{sk},0}$ and $\mathcal{O}^{\mathsf{KDM}}_{\mathcal{F},\mathsf{sk},1}$ are defined in Fig. 3.

Experiment IND-CPA and IND-KDM Security Game

```
$\mathcal{O}^{\mathsf{KDM}}_{\mathcal{F},\mathsf{sk},0}(f)$
if $f \notin \mathcal{F}$ return $\perp$
else
    $\boldsymbol{c} \xleftarrow{\$} \mathsf{Enc}_{\mathsf{pk}}(f(\mathsf{sk}))$
    return $\boldsymbol{c}$
```

```
$\mathcal{O}^{\mathsf{KDM}}_{\mathcal{F},\mathsf{sk},1}(f)$
if $f \notin \mathcal{F}$ return $\perp$
else
    $\boldsymbol{c} \xleftarrow{\$} \mathsf{Enc}_{\mathsf{pk}}(0^{|f(\mathsf{sk})|})$
    return $\boldsymbol{c}$
```

Fig. 3: Oracles used IND-KDM (Definition 5) security.

Note that in all generality, KDM-security requires we allow the plaintexts to be functions of multiple secret keys (*i.e.* $\mathcal{F} \subseteq \mathcal{K}^\ell \to \mathcal{M}$, where ℓ is some parameter, polynomial in the security parameter) and the encryption to be performed under any one of those keys. For our purposes however, it suffices to consider the special case $\ell = 1$. In fact, whenever we use KDM-secure encryption in this paper, it will suffice for the encryption scheme to be secure given encryptions of linear combinations of the key, its inverse, and all constants[6].

3.4 Distributed Discrete Logarithm

We recall in Fig. 4 the definition of distributed discrete logarithm function DDLog introduced by Roy and Singh [RS21]. The properties of this function which we use in this paper are stated in lemmata 6, 7, and 8. Note that these are immediate corollaries/reformulations of [RS21, Theorem 18] and [RS21, Lemma 19]. Throughout, we use the symmetric convention for the mod operator. Namely, "$\cdot \bmod B$" takes values in $(-B/2; B/2]$. We recall in Fig. 4 the definition of distributed discrete logarithm function DDLog introduced by Roy and Singh [RS21]. The properties of this function which we use in this paper are stated in lemmata 6, 7, and 8. Note that these are immediate corollaries/reformulations of [RS21, Theorem 18] and [RS21, Lemma 19]. Throughout, we use the symmetric convention for the mod operator. Namely, the function "$\cdot \bmod B$" takes values in $(-B/2; B/2]$.

DDLOG Damgård-Jurik Distance Function [RS21]

```
$\mathsf{DDLog}(N, h)$ :
    Compute and output $z \leftarrow \log\left(\frac{h}{h \bmod N}\right)$
```

Fig. 4: [RS21]'s distributed discrete logarithm for the Damgård-Jurik cryptosystem [DJ01].

Lemma 6. *For all* $(N, \varphi) \in \mathsf{Supp}(\mathsf{DJ.KeyGen})$*, for all exponents* $\zeta \geq 1$*, and for all ciphertexts* $c \in (\mathbb{Z}/N^{\zeta+1}\mathbb{Z})^\times$ *and shares (over* $\mathbb{Z}$*)* $\langle x\varphi\rangle_0, \langle x\varphi\rangle_1$ *of some* $x \in \mathbb{Z}$ *times* φ*, we have*

$$\mathsf{DDLog}(c^{\langle x\varphi\rangle_1}) - \mathsf{DDLog}(c^{\langle x\varphi\rangle_0}) \equiv \mathsf{DJ.Dec}_{N,\zeta,\varphi}(c) \cdot x \cdot \varphi \mod N^\zeta.$$

Proof. By definition of $\mathsf{DJ.Dec}$ we have $c^\varphi = \exp(\mathsf{DJ.Dec}_{N,\zeta,\varphi}(c) \cdot \varphi)$, so

$$\frac{c^{\langle x\varphi\rangle_1}}{c^{\langle x\varphi\rangle_0}} = c^{x\varphi} = \exp(\mathsf{DJ.Dec}_{N,\zeta,\varphi}(c) \cdot x \cdot \varphi).$$

[6] Note that if $\mathcal{F}$ includes all constant functions from $\mathcal{K}$ to $\mathcal{M}$, then $\mathcal{F}$-KDM security implies CPA security.

That is, the inputs of DDLog are multiplicative shares of $\exp(\mathsf{DJ.Dec}_{N,\zeta,\varphi}(c)\cdot x\cdot\varphi)$. By [RS21, Theorem 18], the outputs of DDLog are then additive secret shares of $\mathsf{DJ.Dec}_{N,\zeta,\varphi}(c)\cdot x\cdot\varphi$. $\square$

Lemma 7. *Let $\zeta \geq 1$. For all $(N,\varphi) \in \mathsf{Supp}(\mathsf{DJ.KeyGen})$, let $\nu = N^{-\zeta} \bmod \varphi$. For all ciphertexts $c \in (\mathbb{Z}/N^{\zeta+1}\mathbb{Z})^\times$, all $x \in \mathbb{Z}$, and all shares $(\langle x\rangle_0, \langle x\rangle_1)$ and $(\langle x\nu\rangle_0, \langle x\nu\rangle_1)$ of x and $x\nu$, we have*

$$\Big[\mathsf{DDLog}(c^{\langle x\rangle_1 - N^\zeta\cdot\langle x\nu\rangle_1}) - \mathsf{DDLog}(c^{\langle x\rangle_0 - N^\zeta\cdot\langle x\nu\rangle_0}) \equiv \mathsf{DJ.Dec}_{N,\zeta,\varphi}(c)\cdot x \mod N^\zeta.$$

Proof. Let $d = (\varphi^{-1} \bmod N^\zeta)\varphi$. We use an observation of [RS21]:

$$1 - N^\zeta\nu \equiv d\varphi \mod N^\zeta\varphi,$$

because they are both 1 modulo N^ζ and they are both 0 modulo φ (and because N and φ are coprime). Now, let $\langle xd\rangle_0 = \langle x\rangle_0 - N^\zeta\cdot\langle x\nu\rangle_0$ and $\langle xd\rangle_1 = \langle xd\rangle_0 + xd$. Then

$$\langle xd\rangle_1 \equiv \langle x\rangle_1 - N^\zeta\cdot\langle x\nu\rangle_1 \mod N^\zeta$$

and the multiplicative order of c divides $N^\zeta\varphi$ by Euler's theorem, so for $i \in \{0,1\}$

$$\mathsf{DDLog}(c^{\langle x\rangle_i - N^\zeta\cdot\langle x\nu\rangle_i}) = \mathsf{DDLog}(c^{\langle xd\rangle_i}).$$

The result then follows from Lemma 6:

$$\mathsf{DJ.Dec}_{N,\zeta,\varphi}(c)\cdot xd\cdot\varphi \equiv \mathsf{DJ.Dec}_{N,\zeta,\varphi}(c)\cdot x\varphi^{-1}\cdot\varphi \equiv \mathsf{DJ.Dec}_{N,\zeta,\varphi}(c)\cdot x \mod N^\zeta. \quad \square$$

Lemma 8. *For all moduli $M > 1$ and all modulo M shares $\langle x\rangle_0, \langle x\rangle_1 \in \mathbb{Z}/M\mathbb{Z}$ of some $x \in \mathbb{Z}$, we have*

$$\Pr_{r \xleftarrow{\$} \mathbb{Z}/M\mathbb{Z}}\Big[(\langle x\rangle_1 + r) \bmod M - (\langle x\rangle_0 + r) \bmod M = x\Big] = \max\left(1 - \frac{|x|}{N^\zeta}, 0\right).$$

Proof. This is essentially [RS21, Lemma 19]. $\square$

4 Arithmetic Garbled Circuits

4.1 From KDM security of Damgård-Jurik

We have already provided a high-level technical overview of this construction in Section 2.2, and we therefore proceed below with the formal description of our garbling scheme (Fig. 5) and its proof of correctness and security.
There is only one difficulty which is not addressed by the technical overview, and this is the reason why our garbling scheme only supports bounded-integer computation, and not full-blown modular arithmetic. By default, computing a DDLog allows the parties to generate shares modulo some value, but these shares need to then be converted into shares over $\mathbb{Z}$ before they can be used as input to another DDLog operation. If the shared value is small enough (which we guarantee by restricting ourselves to B-bounded computation) and the shares are (pseudo)randomised (which we guarantee by having garbler and evaluator offset their shares of each wire value using the same PRF key), then taking these shares modulo B yields correct shares over $\mathbb{Z}$. We note that this trick was previously used in [OSY21].

Arithmetic Garbling AGC from HSS (KDM-secure Damgård-Jurik variant)

Requires:

- $\mathsf{DJ} = (\mathsf{DJ.KeyGen}, \mathsf{DJ.Enc}, \mathsf{DJ.Dec})$ is the Damgård-Jurik cryptosystem of Fig. 2.
- DDLog is the algorithm of Fig. 4.
- $(\mathsf{PRF}_{N,\zeta,\lambda})_{N,\zeta,\lambda\in\mathbb{N}^3}$ is a family of pseudorandom functions s.t. $\mathsf{PRF}_{N,\zeta,\lambda}\colon \{0,1\}^\lambda \times ([N^\zeta] \times \{0,1\}) \to [N^\zeta]$.
- ord is an ordering of the gates of the circuit (*i.e.* a bijection between the set of gates and $[1, s]$, where s is the number of gates).

The Garbling Scheme (gate-by-gate description): Initially, the garbler samples $k_{\mathsf{PRF}} \overset{\$}{\leftarrow} \{0,1\}^\lambda$ and $(N, \varphi) \overset{\$}{\leftarrow} \mathsf{DJ.KeyGen}(1^\lambda)$, sets $k \leftarrow \varphi$, and computes $c \leftarrow \mathsf{DJ.Enc}_N(\varphi^{-1})$ (inversion is performed in $\mathbb{Z}/N^\zeta\mathbb{Z}$); for each input wire x, the garbler samples $K_x \overset{\$}{\leftarrow} [N^\zeta]$, computes $c_x \leftarrow \mathsf{DJ.Enc}_N(K_x)$. The label associated with input x is $L_x \leftarrow k \cdot x + K_x \in [N^\zeta]$.

Addition/Substraction gate $\boldsymbol{z = x \pm y}$: For addition (resp. substraction) gates, at garbling time, given key pairs (k, K_x) and (k, K_y) as well as corresponding ciphertexts c_x and c_y, set $K_z \leftarrow K_x + K_y$ (resp. $K_z \leftarrow K_x - K_y$), compute the ciphertext $c_z \leftarrow c_x \cdot c_y$ (resp. $c_z \leftarrow c_x/c_y$) (c_z will be stored in order to compute later keys and tables, but is not included in the garbled circuit), and produce the key pair (k, K_z). At evaluation time, given the labels L_x and L_y, compute the label $L_z \leftarrow L_x + L_y$ (resp. $L_z \leftarrow L_x - L_y$) and the ciphertext $c_z \leftarrow c_x \cdot c_y$ (resp. $c_z \leftarrow c_x/c_y$).

Multiplication gate $\boldsymbol{z = x \cdot y}$: At garbling time, given key pairs (k, K_x) and (k, K_y) as well as the corresponding ciphertexts c_x and c_y, compute the key pair (k, K_z), where K_z is computed as follows:

$$\mathsf{sh}_z^G \leftarrow -\mathsf{DDLog}(c_x^{K_y}) - \mathsf{DDLog}(c_y^{K_x}) - K_x \cdot K_y + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, \mathsf{ord}(z)\|0) \bmod N^\zeta$$
$$K_z \leftarrow \mathsf{DDLog}(c^{\mathsf{sh}_z^G}) + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, \mathsf{ord}(z)\|1) \bmod N^\zeta$$

Then produce the ciphertext $c_z \overset{\$}{\leftarrow} \mathsf{Enc}_N(K_z)$ (this ciphertext will be included in the garbled circuit). At evaluation time, given labels L_x and L_y as well as the corresponding ciphertexts c_x and c_y, compute the label L_z as follows:

$$\mathsf{sh}_z^E \leftarrow L_x \cdot L_y - \mathsf{DDLog}(c_x^{L_y}) - \mathsf{DDLog}(c_y^{L_x}) + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, \mathsf{ord}(z)\|0) \bmod N^\zeta$$
$$L_z \leftarrow \mathsf{DDLog}(c^{\mathsf{sh}_z^E}) + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, \mathsf{ord}(z)\|1) \bmod N^\zeta$$

Output: For each output z, the garbler computes $\mathsf{out}_z^G \leftarrow \mathsf{DDLog}(c^{K_z}) + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, \mathsf{ord}(z)\|2) \bmod N^\zeta$ and outputs:

- The garbled circuit including: N, c, c_z for each mult. z, and out_z^G for each output z.
- The input encoding information: k, and K_x for each input x.

The evaluator runs $\mathsf{out}_z^E \leftarrow \mathsf{DDLog}(c^{L_z}) + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, \mathsf{ord}(z)\|3) \bmod N^\zeta$ and outputs $(\mathsf{out}_z^E - \mathsf{out}_z^G)$ for each output z.

Fig. 5: Arithmetic garbled circuit with one ciphertext per multiplication (and free addition) from KDM-security of Damgård-Jurik (gate-by-gate description).

Theorem 9 (AGC with one ciphertext per multiplication from KDM-secure Damgård-Jurik, via DDLog). *Let λ be a security parameter. Let $\zeta \geq 2$, assume the KDM-security*[7] *of the Damgård-Jurik encryption scheme with parameter ζ, and let $\ell(\lambda)$ be the bit-length of the corresponding RSA modulus. Let $\varepsilon > 0$. For every $B \leq 2^{(\zeta-2-\varepsilon)\cdot\ell(\lambda)}$, the construction of Fig. 5 is an arithmetic garbling scheme for B-bounded integer computation. Moreover, if a circuit C has n inputs, m outputs, and $s_\times$ multiplication gates then its garbled circuit has bit-size*

$$(n + s_\times + 1) \cdot \underbrace{(\zeta + 1) \cdot \ell(\lambda)}_{\substack{\textit{size of a DJ} \\ \textit{ciphertext}}} \quad + \quad \underbrace{m \cdot \zeta \cdot \ell(\lambda)}_{\textit{decoding material}}$$

while each input label is $2\zeta \cdot \ell(\lambda)$ bits.

Proof.

- ***Correctness.*** Let us show that the following invariant is maintained with all but negligible probability (w.a.b.n.p.) for each wire w of the circuit: $L_w = k \cdot w + K_w$. Let us show (by a case analysis) that for each gate of the circuit, if the invariant holds for the gate's input wires then it holds w.a.b.n.p. for its output wires.

  - If w is an input wire, then the invariant holds by definition of L_w.
  - If $w = x \pm y$, and the invariant holds for x and y, then by definition $L_w \overset{\mathsf{def}}{=} L_x \pm L_y$ and $K_w \overset{\mathsf{def}}{=} K_x \pm K_y$. Therefore $L_w = (k \cdot x + K_x) + (k \cdot y + K_y) = k \cdot (x + y) + (K_x + K_y) = k \cdot w + K_w$.
  - If $w = x \cdot y$, assume the invariant holds for x and y. By correctness of the Damgård-Jurik cryptosystem, $\mathsf{DJ.Dec}(c_x) = K_x$ w.a.b.n.p. By Lemma 6, because $L_y - K_y = y \cdot \varphi$, $\mathsf{DDLog}(c_x^{L_y}) - \mathsf{DDLog}(c_x^{K_y}) = (\varphi \cdot yK_x) \bmod N^\zeta$ w.a.b.n.p. Similarly, $\mathsf{DDLog}(c_y^{L_x}) - \mathsf{DDLog}(c_y^{K_x}) = (\varphi \cdot xK_y) \bmod N^\zeta$ w.a.b.n.p. Therefore,

    $$\mathsf{sh}_w^E - \mathsf{sh}_w^E =_{\mathbb{Z}} L_x \cdot L_w - (\varphi \cdot yK_x \bmod N^\zeta) - (\varphi \cdot xK_y \bmod N^\zeta) - K_x \cdot K_y.$$

    By Eq. (2) (and because $L_x = \varphi x + K_x$ and $L_y = \varphi y + K_y$),

    $$\mathsf{sh}_w^E - \mathsf{sh}_w^E \equiv \varphi^2 \cdot xy \equiv \varphi^2 \cdot w \bmod N^\zeta. \qquad (4)$$

[7] More precisely, the theorem assumes KDM-security with respect to the class containing the following functions: the inverse function, all linear combinations, and all constant functions. Note that because the Damgård-Jurik cryptosystem is linearly homomorphic, linear combination are well-defined.

Lemma 10. *Over the randomness of* $(k, (K_{x_i})_{i\in[n]}, \widehat{C}) \xleftarrow{\$} \mathsf{Garble}(1^\lambda, C)$; $L_{x_i} \leftarrow k \cdot x_i + K_{x_i}$,

$$\Pr[\mathsf{sh}_w^E - \mathsf{sh}_w^G \neq \varphi^2 \cdot w] \leq 1/N^\varepsilon + \mathsf{negl}(\lambda) \ .$$

Proof of Lemma 10. Let $r = \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, \mathsf{ord}(z))$, and obeserve that k_{PRF} is sampled uniformly and only used to generate these r values. Therefore, we can use PRF security to replace r with a uniformly random value in $\mathbb{Z}/N^\zeta\mathbb{Z}$, and only add a negligible term to our bound. (The inputs are chosen before the garbled circuit is output, so it does not matter that k_{PRF} is given to the evaluator as part of the garbled circuit.)
Next, we want to bound the probability of $\mathsf{sh}_w^E - \mathsf{sh}_w^G \neq \varphi^2 \cdot w$. Note that $\mathsf{sh}_w^E = \widetilde{\mathsf{sh}}_w^E + r$ and $\mathsf{sh}_w^G = \widetilde{\mathsf{sh}}_w^G + r$ for some shares $\widetilde{\mathsf{sh}}_w^E, \widetilde{\mathsf{sh}}_w^G$ independent of r. Recall from Eq. (4), sh_w^E and sh_w^G are shares of $\phi^2 \cdot w$ modulo N^ζ. It follows that so are $\widetilde{\mathsf{sh}}_w^E$ and $\widetilde{\mathsf{sh}}_w^G$. By Lemma 8,

$$\Pr_{r \xleftarrow{\$} \mathbb{Z}/N^\zeta\mathbb{Z}}[(\widetilde{\mathsf{sh}}_w^E + r) \bmod N^\zeta - (\widetilde{\mathsf{sh}}_w^E + r) \bmod N^\zeta \neq \varphi^2 \cdot w] \leq \frac{|\varphi^2 \cdot w|}{N^\zeta} \ .$$

By B-admissibility, $|w| \leq B$, and since $B \leq N^{\zeta-2-\varepsilon}$ by assumption, $\frac{|\varphi^2 \cdot w|}{N^\zeta} \leq 1/N^\varepsilon$. □

We now resume the proof. By applying Lemma 10, $\mathsf{sh}_w^E - \mathsf{sh}_w^G = \varphi^2 \cdot w$ w.a.b.n.p. By correctness of the Damgård-Jurik cryptosystem, $\mathsf{DJ.Dec}(c) = \varphi^{-1}$ w.a.b.n.p. and by Lemma 6 $\mathsf{DDLog}(\mathsf{sh}_w^E) - \mathsf{DDLog}(\mathsf{sh}_w^G) \equiv \varphi^{-1} w \varphi \bmod N^\zeta \equiv w \bmod N^\zeta$. Completely analogously to Lemma 10, it holds that $L_w - K_w = \varphi \cdot w$ w.a.b.n.p. Because there are only a polynomial number of gates, and for each one the invariant holds w.a.b.n.p. if it holds for the gate's input wires, with all but negligible probability the invariant holds for every wire of the circuit. In particular, for every output value z, $L_z - K_z = \varphi \cdot z$. Because the inputs of the circuit are B admissible, $\varphi \cdot z \leq N^\zeta$. By Lemma 6 (and by correctness of the Damgård-Jurik cryptosystem), $\mathsf{out}_z^E - \mathsf{out}_z^G$

– ***Privacy.*** Consider the following simulator $\mathcal{S}$ which, on input $(1^\lambda, C_\lambda, \boldsymbol{y} = (y_1, \dots, y_m))$, does the following: (1) sample $k_{\mathsf{PRF}} \xleftarrow{\$} \{0,1\}^\lambda$ and $(N, \varphi) \xleftarrow{\$} \mathsf{DJ.KeyGen}(1^\lambda)$, (2) sample $c \xleftarrow{\$} \mathsf{DJ.Enc}_N(0)$ and, for each input or multiplication z of C_λ, sample $c_z \xleftarrow{\$} \mathsf{DJ.Enc}_N(0)$, (3) if $w_1, \dots, w_m$ are the outputs of C_λ, sample $\mathsf{out}_w^E \xleftarrow{\$} (-N^\zeta/2, N_D^\zeta/2]$ then set $\mathsf{out}_{w_i}^G \leftarrow \mathsf{out}_{w_i}^E - y_i$, (4) for each input x of C_λ, sample $L_x \xleftarrow{\$} [N^\zeta]$, (5) set $\widehat{C}_\lambda \leftarrow (N, c, (c_z)_z, (\mathsf{out}_w^G)_w)$, and output $((L_x)_x, \widehat{C}_\lambda)$.
Let us now show, via a hybrid argument, that this simulator satisfies the privacy requirement of Definition 1.
  1. Starting from the real-world experiment, for each output wire w_i we replace $\mathsf{out}_{w_i}^G$ with $\mathsf{out}_{w_i}^E - y_i$. These two hybrids are statistically indistinguishable by correctness of the garbling scheme (established earlier in this proof).
  2. We now replace ciphertexts given in the garbled circuit with encryptions of 0. This new hybrid (which is also the ideal world) is indistinguishable from the previous by KDM security of the Damgård-Jurik cryptosystem. In fact, the two distributions are indistinguishable even if the inputs to the garbled circuit are made public. Indeed, otherwise, an adversary tasked with distinguishing $\mathcal{O}_{\mathcal{F},\mathsf{sk},0}^{\mathsf{KDM}}$ and $\mathcal{O}_{\mathcal{F},\mathsf{sk},1}^{\mathsf{KDM}}$ could run the experiment of sampling k_{PRF}, L_x for each input x, and proceed to compute labels in a gate-by-gate fashion. Whenever the reduction requires a ciphertext in order to then perform a DDLog operation, it queries the oracle:

- To obtain c (which, in the real world, should be an encryption of the inverse of the key), it queries the oracle on $f\colon X \mapsto X^{-1}$.
- To obtain c_w (which, in the real world, should be an encryption of K_w) for some wire x for which it already knows the label L_w, it queries the oracle on $f\colon -w \cdot X + L_w$. By correctness, of the garbling scheme, $K_w = -w \cdot \varphi + L_w = f(\varphi)$. Note that since we assumed the inputs to the circuit were public, the reduction can indeed compute the value of wire w in the clear.

Observe that this experiment corresponds exactly to either the first hybrid (if the oracle is $\mathcal{O}^{\mathsf{KDM}}_{\mathcal{F},\mathsf{sk},0}$) or the second one (if the oracle is $\mathcal{O}^{\mathsf{KDM}}_{\mathcal{F},\mathsf{sk},1}$). It follows that if the two hybrids were not indistinguishable, one could therefore distinguish between oracle access to $\mathcal{O}^{\mathsf{KDM}}_{\mathcal{F},\mathsf{sk},0}$ and $\mathcal{O}^{\mathsf{KDM}}_{\mathcal{F},\mathsf{sk},1}$, which would violate KDM security.

– ***Size.*** The garbled circuit is comprised of $(n + s_\times + 1)$ Damgård-Jurik ciphertexts (c_x for each input x, c_z for each multiplication z, and c), as well as the garbler's share of each output (out^G_z for each output z).

□

4.2 From CPA security of Damgård-Jurik

In Section 4.2.1 we descibe (in a "overview" style, introducing one technique at a time) the techniques which allow us to remove the need for KDM security assumptions. We refer to Section 4.2.2 for a formal treatment, with the resulting construction described in Fig. 7.

4.2.1 Techniques to remove circular security. As discussed in Section 2.4, the main idea behind removing the dependence on KDM-style assumptions is to use the powerful idea of *key-switching i.e.*, for each layer of multiplications in the circuit the garbled will generate a different Damgård-Jurik key-pair, and only encrypt (functions of) the secret-key in layer $i+1$ under the public key of layer i.
Armed with this idea, there are still a few technical issues to address before we are ready to provide our formal construction. Also, for the construction in this section, keys and labels are defined in such a way as to maintain the invariant

$$\boldsymbol{L}_w = (1, d_i) \cdot w - \boldsymbol{K}_w,$$

where i is the multiplicative depth of wire w.

For each addition gate $z = x + y$, where x, y, z are at depth i_1, i_2, j (with $0 \le i_1 < j \le D-1$, $0 \le i_2 < j \le D-1$), the parties (1) decompose their labels/keys for x and y as $\langle x \rangle \| \langle d_{i_1} \cdot x \rangle$ and $\langle y \rangle \| \langle d_{i_2} \cdot y \rangle$, (2) perform key-switches to generate $\langle d_j \cdot x \rangle$ and $\langle d_j \cdot y \rangle$, (3) set their label/key for z to be $(\langle x \rangle + \langle y \rangle) \| (\langle d_j \cdot x \rangle + \langle d_j \cdot y \rangle)$. By linearity, these keys and labels indeed satisfy $\boldsymbol{L}_z = (1, d_j) \cdot z - \boldsymbol{K}_z$.

For each multiplication gate $z = x \cdot y$, where x, y, z are at depth i_1, i_2, j (with $0 \le i_1 < j \le D-1$, $0 \le i_2 < j \le D-1$), we exploit the identity from Eq. (3) restated here for convenience:

$$z = \langle x \rangle_1 \cdot \langle y \rangle_1 - x \cdot \langle y \rangle_0 - y \cdot \langle x \rangle_0 - \langle x \rangle_0 \cdot \langle y \rangle_0 \tag{5}$$

1. *Generate* $\langle z \rangle$. The garbler includes in the garbled circuit encryptions of its shares under the public key of the final layer D i.e., ciphertexts $c_x \xleftarrow{\$} \mathsf{Enc}_{N_D}(\langle x \rangle_0)$ and $c_y \xleftarrow{\$} \mathsf{Enc}_{N_D}(\langle y \rangle_0)$, and the parties (1) perform key switches to generate $\langle d_D \cdot x \rangle$ and $\langle d_D \cdot y \rangle$, (2) compute $\langle x \cdot \langle y \rangle_0 \rangle \leftarrow \mathsf{DDLog}(c_y^{\langle d_D \cdot x \rangle})$ and $\langle y \cdot \langle x \rangle_0 \rangle \leftarrow \mathsf{DDLog}(c_x^{\langle d_D \cdot y \rangle})$, (3) use Eq. (5) to compute shares of z.

2. *Generate* $\langle d_j \cdot z\rangle$. Assume that for $i \in [0, D-1]$, the garbler provided ciphertexts $c'_i \leftarrow \mathsf{Enc}_{N_i}(d_{i+1}^2)$ (these ciphertexts are to be globally reused for every multiplication gate). The garbler performs key-switches to convert $\langle d_{i_1} \cdot x\rangle_0$ to $\langle d_j \cdot x\rangle_0$, and then additionally provides the ciphertext $\mathsf{ct}_x \xleftarrow{\$} \mathsf{Enc}_{N_j}(\langle d_j \cdot x\rangle_0)$. The parties do the following:
   (a) perform key switching to compute $\langle d_{j-1} \cdot x\rangle$, $\langle d_j \cdot x\rangle$, $\langle d_{j-1} \cdot y\rangle$, and $\langle d_j \cdot y\rangle$;
   (b) compute $\langle (d_j)^2 \cdot x\rangle \leftarrow \mathsf{DDLog}((c'_j)^{\langle d_{j-1}\cdot x\rangle})$ and $\langle (d_j)^2 \cdot y\rangle \leftarrow \mathsf{DDLog}((c'_j)^{\langle d_{j-1}\cdot y\rangle})$;
   (c) perform $(j - i_1)$ key-switches using $(c_i)_{i=i_1}^{j}$ to convert $\langle (d_j)^2 \cdot x\rangle$ to $\langle d_D \cdot d_j \cdot x\rangle$;[8]
   (d) perform $(j - i_2)$ key-switches using $(c_i)_{i=i_2}^{j}$ to convert $\langle (d_j)^2 \cdot y\rangle$ to $\langle d_D \cdot d_j \cdot y\rangle$;
   (e) compute $\langle d_j x \cdot \langle y\rangle_0\rangle \leftarrow \mathsf{DDLog}(c_y^{\langle d_D \cdot d_j x\rangle})$ and $\langle y \cdot \langle d_j \cdot x\rangle_0\rangle \leftarrow \mathsf{DDLog}(\mathsf{ct}_x^{\langle d_D \cdot y\rangle})$;
   (f) use the identity of Eq. (6), $\langle x\rangle$, $\langle y\rangle$, $\langle d_j x \cdot \langle y\rangle_0\rangle$, and $\langle y \cdot \langle d_j \cdot x\rangle_0\rangle$ to compute $\langle d_j \cdot z\rangle$.

$$d_j \cdot z = \langle d_j \cdot x\rangle_1 \cdot \langle y\rangle_1 - d_j x \cdot \langle y\rangle_0 - y \cdot \langle d_j \cdot x\rangle_0 - \langle d_j \cdot x\rangle_0 \cdot \langle y\rangle_0 \tag{6}$$

With this approach, the garbled circuit contains two ciphertext per multiplication gate (that is, c_x and ct_z for $z = x \cdot y$), plus an additionnal $2D$ ciphertexts used to perform key switches (that is, the encryptions of d_{i+1} and d_{i+1}^2 under (N_i, sk_i) for $i \in [0, D-1]$).
There is a problem however with the scheme as presented. Because the d_i can, in all generality, be as large as N^ζ, and we need authenticated shares of the wire values to fit in the plaintext space, it seems that we can only performe arithmetic over $\mathbb{F}_2$. Fortunately, there is a simple fix to this last problem.

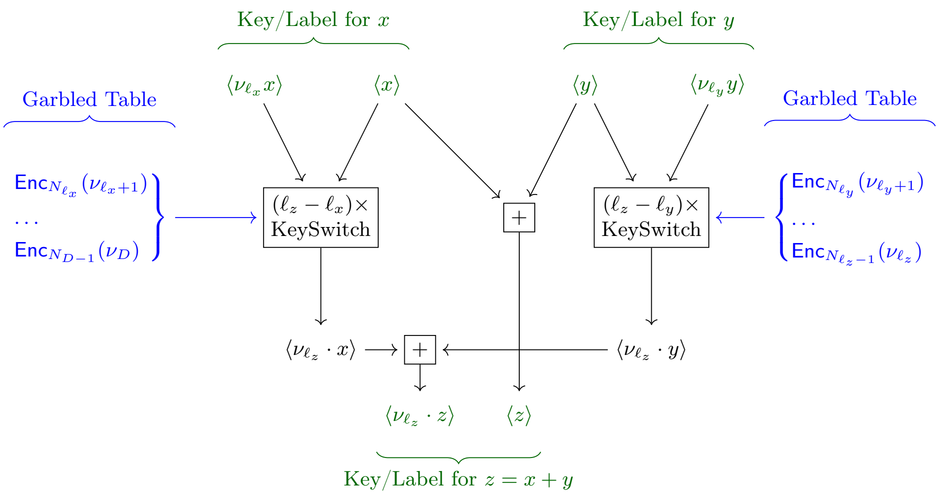
Replace $\boldsymbol{\langle d \cdot X\rangle}$ ***with*** $\boldsymbol{\langle \nu \cdot X\rangle}$. Roy and Singh [RS21, Section 4.3] introduced a trick in order to bypass the above problem. If (N, sk) is a keypair for the Damgård-Jurik encryption scheme, and we define $d \leftarrow \mathsf{sk} \cdot (\mathsf{sk}^{-1} \bmod N^\zeta)$ and $\nu \leftarrow N^{-\zeta} \bmod \mathsf{sk}$, then by the Chinese remainders theorem $d \equiv 1 - N^\zeta \nu \bmod \mathsf{sk} \cdot N^\zeta$. Since N^ζ is public, this allows parties to locally convert ν-authenticated shares into d-authenticated ones: $\langle d \cdot X\rangle \leftarrow 1 - N^\zeta \cdot \langle \nu \cdot X\rangle$. Crucially, since $\nu \leq \mathsf{sk} \leq N$, we can encrypt ν-authenticated shares of wire values as large as $N^{\zeta-1}$. In turn the garbling scheme has rate $(\zeta - 1)/(2\zeta + 2)$ (assuming that the multiplicative depth of the circuit is significantly smaller than its size).
Concretely, key-switches are now performed as follows, given $\langle x\rangle$, $\langle \nu \cdot x\rangle$, $c_i \stackrel{\mathsf{def}}{=} \mathsf{Enc}_{N_i}(\nu_{i+1})$:

1. compute $\langle d_i \cdot x\rangle \leftarrow \langle x\rangle - N_i^\zeta \langle \nu \cdot x\rangle$;
2. compute $\langle \nu_{i+1} \cdot x\rangle \leftarrow \mathsf{DDLog}(c_i^{\langle d_i \cdot x\rangle})$;
3. optionally, use $\langle x\rangle$ again to compute $\langle d_{i+1} \cdot x\rangle$ and continue key-switching recursively.

We summarise in Figs. 6a and 6b how garbler and evaluator now perform additions and multiplications.

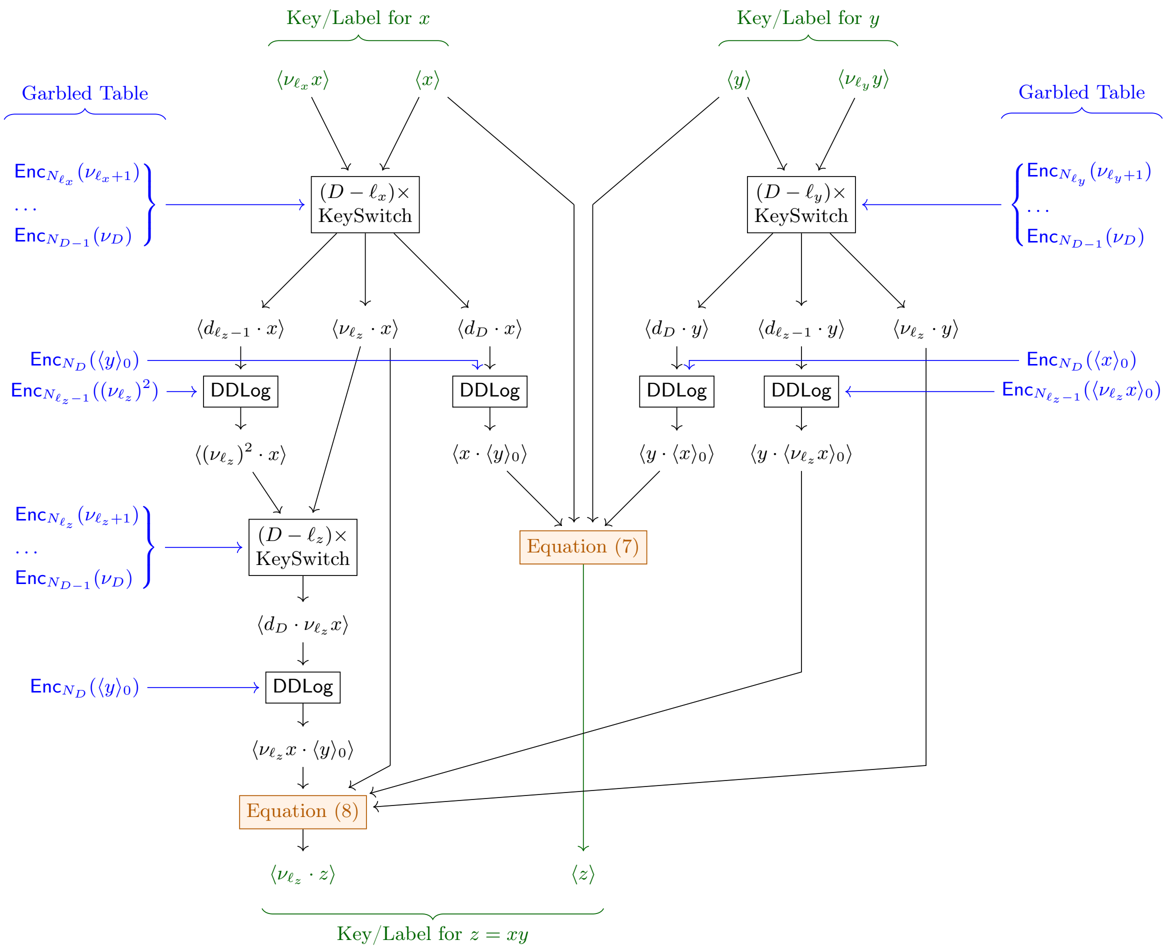
[8] Note that $\langle (d_j)^2 \cdot x\rangle$ can be seen as $\langle d_j \cdot (d_j \cdot x)\rangle$: the leading d_j is the one being "switched out for d_D".

Fig. 6: Operations performed by the garbler (resp. evaluator) to convert their keys (resp. labels) for wires x (at multiplicative depth ℓ_x) and y (at multiplicative depth ℓ_y) into a key (resp. label) for wire $z = f(x, y)$ (at multiplicative depth ℓ_z), where $f \in \{+, \times\}$. This only relies on CPA-security of Damgård-Jurik encryption.



Key/Label for x
Key/Label for y
Garbled Table
Garbled Table
$\langle \nu_{\ell_x} x \rangle$
$\langle x \rangle$
$\langle y \rangle$
$\langle \nu_{\ell_y} y \rangle$
$\mathsf{Enc}_{N_{\ell_x}}(\nu_{\ell_x+1})$ … $\mathsf{Enc}_{N_{D-1}}(\nu_D)$
$(\ell_z - \ell_x)\times$ KeySwitch
$+$
$(\ell_z - \ell_y)\times$ KeySwitch
$\mathsf{Enc}_{N_{\ell_y}}(\nu_{\ell_y+1})$ … $\mathsf{Enc}_{N_{\ell_z-1}}(\nu_{\ell_z})$
$\langle \nu_{\ell_z} \cdot x \rangle$
$+$
$\langle \nu_{\ell_z} \cdot y \rangle$
$\langle \nu_{\ell_z} \cdot z \rangle$
$\langle z \rangle$
Key/Label for $z = x + y$

(a) Addition gates.



Key/Label for x
Key/Label for y
Garbled Table
Garbled Table
$\langle \nu_{\ell_x} x \rangle$
$\langle x \rangle$
$\langle y \rangle$
$\langle \nu_{\ell_y} y \rangle$
$\mathsf{Enc}_{N_{\ell_x}}(\nu_{\ell_x+1})$ … $\mathsf{Enc}_{N_{D-1}}(\nu_D)$
$(D - \ell_x)\times$ KeySwitch
$(D - \ell_y)\times$ KeySwitch
$\mathsf{Enc}_{N_{\ell_y}}(\nu_{\ell_y+1})$ … $\mathsf{Enc}_{N_{D-1}}(\nu_D)$
$\langle d_{\ell_z-1} \cdot x \rangle$
$\langle \nu_{\ell_z} \cdot x \rangle$
$\langle d_D \cdot x \rangle$
$\langle d_D \cdot y \rangle$
$\langle d_{\ell_z-1} \cdot y \rangle$
$\langle \nu_{\ell_z} \cdot y \rangle$
$\mathsf{Enc}_{N_D}(\langle y \rangle_0)$
$\mathsf{Enc}_{N_{\ell_z-1}}((\nu_{\ell_z})^2)$
DDLog
DDLog
DDLog
DDLog
$\mathsf{Enc}_{N_D}(\langle x \rangle_0)$
$\mathsf{Enc}_{N_{\ell_z-1}}(\langle \nu_{\ell_z} x \rangle_0)$
$\langle (\nu_{\ell_z})^2 \cdot x \rangle$
$\langle x \cdot \langle y \rangle_0 \rangle$
$\langle y \cdot \langle x \rangle_0 \rangle$
$\langle y \cdot \langle \nu_{\ell_z} x \rangle_0 \rangle$
$\mathsf{Enc}_{N_{\ell_z}}(\nu_{\ell_z+1})$ … $\mathsf{Enc}_{N_{D-1}}(\nu_D)$
$(D - \ell_z)\times$ KeySwitch
Equation (7)
$\langle d_D \cdot \nu_{\ell_z} x \rangle$
$\mathsf{Enc}_{N_D}(\langle y \rangle_0)$
DDLog
$\langle \nu_{\ell_z} x \cdot \langle y \rangle_0 \rangle$
Equation (8)
$\langle \nu_{\ell_z} \cdot z \rangle$
$\langle z \rangle$
Key/Label for $z = xy$

Key Identities

$$z = \langle x \rangle_1 \cdot \langle y \rangle_1 - x \cdot \langle y \rangle_0 - y \cdot \langle x \rangle_0 - \langle x \rangle_0 \cdot \langle y \rangle_0 \tag{7}$$

$$\nu_{\ell_z} \cdot z = \langle \nu_{\ell_z} \cdot x \rangle_1 \cdot \langle y \rangle_1 - \nu_{\ell_z} x \cdot \langle y \rangle_0 - y \cdot \langle \nu_{\ell_z} \cdot x \rangle_0 - \langle \nu_{\ell_z} \cdot x \rangle_0 \cdot \langle y \rangle_0 \tag{8}$$

(b) Multiplication gates.

4.2.2 Formal construction, theorem statement, and proof.

Arithmetic Garbling AGC from HSS (CPA-secure Damgård-Jurik variant)

Requires:

- $\mathsf{DJ} = (\mathsf{DJ.KeyGen}, \mathsf{DJ.Enc}, \mathsf{DJ.Dec})$ is the Damgård-Jurik cryptosystem of Fig. 2.
- DDLog is the algorithm of Fig. 4.
- $(\mathsf{PRF}_{N,\zeta,\lambda})_{N,\zeta,\lambda \in \mathbb{N}^3}$ is a family of pseudorandom functions with outputs in $[N^\zeta]$.
- ord is an ordering of the gates of the circuit (*i.e.* a bijection between the set of gates and $[1, s]$, where s is the number of gates).

The Garbling Scheme (gate-by-gate description): Initially, the garbler samples $k_{\mathsf{PRF}} \xleftarrow{\$} \{0,1\}^\lambda$ and $(N_\ell, \varphi_\ell) \xleftarrow{\$} \mathsf{DJ.KeyGen}(1^\lambda)$ then sets $\nu_\ell \leftarrow N^{-\zeta} \bmod \varphi_\ell$ for each $\ell \in [0, D]$ (where D is the multiplicative depth of the circuit). For each $\ell \in [0, D-1]$, the garbler computes $c_\ell \leftarrow \mathsf{DJ.Enc}_{N_\ell}(\nu_{\ell+1})$ and $c'_\ell \leftarrow \mathsf{DJ.Enc}_{N_\ell}(\nu_{\ell+1}^2)$; for each input wire x, the garbler samples $\boldsymbol{K}_x \xleftarrow{\$} [N_0^s]^2$. The label associated with input x is $\boldsymbol{L}_x \leftarrow (1, \nu_0) \cdot x + \boldsymbol{K}_x \in [N_0^s]^2$.

Addition gate $\boldsymbol{z = x + y}$: Let $\ell_x, \ell_y, \ell_z \in [0, D]$ be the multiplicative depths of wires x, y, and z respectively (and observe that $\ell_z = \max(\ell_x, \ell_y)$). Let w, W such that $\{w, W\} = \{x, y\}$, $\ell_w = \min(\ell_x, \ell_y)$ and $\ell_W = \max(\ell_x, \ell_y)$.

Addition – Garbling

At garbling time, given $((1, \nu_{\ell_x}), \boldsymbol{K}_x)$ and $((1, \nu_{\ell_y}), \boldsymbol{K}_y)$ as well as corresponding ciphertexts c_x and c_y:

1. **Key switching to lift $\boldsymbol{\langle d_{\ell_w} \cdot w \rangle}$ to $\boldsymbol{\langle d_{\ell_z} \cdot w \rangle}$.** Set $\langle w \rangle_0 \leftarrow \boldsymbol{K}_w.\mathsf{fst}$ and $\langle \nu_{d_w} \cdot w \rangle_0 \leftarrow \boldsymbol{K}_w.\mathsf{snd}$, then, for each $\ell \in [\ell_w, \ell_z - 1]$, compute $\langle \nu_{\ell+1} \cdot w \rangle_0$ and $\langle d_\ell \cdot w \rangle_0$ as follows:
$$\langle d_\ell \cdot w \rangle_0 \leftarrow \langle w \rangle_0 - N_\ell^s \cdot \langle \nu_\ell \cdot w \rangle_0$$
$$\langle \nu_{\ell+1} \cdot w \rangle_0 \leftarrow \mathsf{DDLog}((c_\ell)^{\langle d_\ell \cdot w \rangle_0}) + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, (\mathsf{ord}(w), 0, \ell, 1)) \bmod N^\zeta\ .$$
2. Homomorphically compute the ciphertext $c_z \leftarrow c_x \cdot c_y$ (to be stored, but not given out as part of the garbled table).
3. Compute $\boldsymbol{K}_z \leftarrow (\boldsymbol{K}_w.\mathsf{fst} + \boldsymbol{K}_W.\mathsf{fst},\ \langle \nu_{\ell_z} \cdot w \rangle_0 + \boldsymbol{K}_W.\mathsf{snd})$ and output $(\nu_{\ell_z}, \boldsymbol{K}_z)$.

Addition – Evaluation

At evaluation time, given labels $\boldsymbol{L}_x$ and $\boldsymbol{L}_y$, compute the label $\boldsymbol{L}_z$ as follows:

1. **Key switching to lift $\boldsymbol{\langle d_{\ell_w} \cdot w \rangle}$ to $\boldsymbol{\langle d_{\ell_z} \cdot w \rangle}$.** Set $\langle w \rangle_1 \leftarrow \boldsymbol{L}_w.\mathsf{fst}$ and $\langle \nu_{\ell_w} \cdot w \rangle_1 \leftarrow \boldsymbol{L}_w.\mathsf{snd}$, then, for each $\ell \in [\ell_w, \ell_z - 1]$, compute $\langle \nu_{\ell+1} \cdot w \rangle_1$ and $\langle d_\ell \cdot w \rangle_1$ as follows:
$$\langle d_\ell \cdot w \rangle_1 \leftarrow \langle w \rangle_1 - N_\ell^s \cdot \langle \nu_\ell \cdot w \rangle_1$$
$$\langle \nu_{\ell+1} \cdot w \rangle_1 \leftarrow \mathsf{DDLog}((c_\ell)^{\langle d_\ell \cdot w \rangle_1}) + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, (\mathsf{ord}(w), 0, \ell, 1)) \bmod N^\zeta\ .$$
2. Compute and store $c_z \leftarrow c_x \cdot c_y$.
3. Compute and output the label $\boldsymbol{L}_z \leftarrow (\boldsymbol{L}_w.\mathsf{fst} + \boldsymbol{L}_W.\mathsf{fst},\ \langle \nu_{\ell_z} \cdot w \rangle_1 + \boldsymbol{L}_W.\mathsf{snd})$.

Multiplication gate $z = x \cdot y$: Let $\ell_x, \ell_y, \ell_z \in [0, D]$ be the multiplicative depths of wires x, y, and z respectively (and observe that $\ell_z = \max(\ell_x, \ell_y) + 1$).

Multiplication – Garbling

At garbling time, given $((1, \nu_{\ell_x}), \boldsymbol{K}_x)$ and $((1, \nu_{\ell_y}), \boldsymbol{K}_y)$, as well as corresponding ciphertexts c_x and c_y, compute $\boldsymbol{K}_z$ as follows:

1. For $w \in \{x, y\}$, set $\langle w \rangle_0 \leftarrow \boldsymbol{K}_w.\mathsf{fst}$ and $\langle \nu_{\ell_w} \cdot w \rangle_0 \leftarrow \boldsymbol{K}_w.\mathsf{snd}$, then, for each $\ell \in [\ell_w, D-1]$, compute $\langle \nu_{\ell+1} \cdot w \rangle_0$ and $\langle d_\ell \cdot w \rangle_0$ as follows:

$$\begin{aligned} \langle d_\ell \cdot w \rangle_0 &\leftarrow \langle w \rangle_0 - N_\ell^s \cdot \langle \nu_\ell \cdot w \rangle_0 \\ \langle \nu_{\ell+1} \cdot w \rangle_0 &\leftarrow \mathsf{DDLog}((c_\ell)^{\langle d_\ell \cdot w \rangle_0}) + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, (\mathsf{ord}(w), 1, \ell)) \bmod N^\zeta \ . \end{aligned}$$

   Then, compute $\mathsf{ct}_x \xleftarrow{\$} \mathsf{DJ.Enc}_{N_{\ell_z - 1}}(\langle \nu_{\ell_z} \cdot x \rangle_0)$.
2. Compute $\langle z \rangle_0$ as follows:

$$\begin{aligned} \langle z \rangle_0 \leftarrow \big(&- \mathsf{DDLog}((c_x)^{\langle d_D \cdot y \rangle_0}) - \mathsf{DDLog}((c_y)^{\langle d_D \cdot x \rangle_0}) - \boldsymbol{K}_x.\mathsf{fst} \cdot \boldsymbol{K}_y.\mathsf{fst} \\ &+ \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, (\mathsf{ord}(w), 2)) \big) \bmod N^\zeta \ . \end{aligned}$$

3. Compute $\langle \nu_{\ell_z}^2 \cdot x \rangle_0 \leftarrow \mathsf{DDLog}((c'_{\ell_z - 1})^{\langle d_{\ell_z - 1} \cdot x \rangle_0}) + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, (\mathsf{ord}(w), 3)) \bmod N^\zeta$ then compute $(\langle d_\ell \cdot \nu_{\ell_z} \cdot x \rangle_0)_{\ell = \ell_z + 1}^D$ recursively as:

$$\langle d_{\ell_z} \cdot \nu_{\ell_z} \cdot x \rangle_0 \leftarrow \langle \nu_{\ell_z} \cdot x \rangle_0 - N_\ell^s \cdot \langle \nu_{\ell_z}^2 \cdot x \rangle_0$$

   For $\ell \in [\ell_z, D-1]$, $\langle d_{\ell+1} \cdot \nu_{\ell_z} \cdot x \rangle_0 \leftarrow \big(\langle \nu_{\ell_z} \cdot x \rangle_0 - N_\ell^s \cdot \mathsf{DDLog}((c'_\ell)^{\langle d_\ell \cdot \nu_{\ell_z} \cdot x \rangle_0})$
$$+ \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, (\mathsf{ord}(w), 4, \ell)) \bmod N^\zeta \big) \ .$$

   Compute:

$$\begin{aligned} \langle \nu_{\ell_z} \cdot z \rangle_0 \leftarrow \big(&- \mathsf{DDLog}(c_y^{\langle d_D \cdot \nu_{\ell_z} \cdot x \rangle_0}) - \mathsf{DDLog}(\mathsf{ct}_x^{\langle d_{\ell_z - 1} \cdot y \rangle_0}) - \langle \nu_{\ell_z} \cdot x \rangle_0 \cdot (\boldsymbol{K}_y.\mathsf{fst}) \\ &+ \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, (\mathsf{ord}(w), 5)) \bmod N^\zeta \big) \ . \end{aligned}$$

   Finally, compute $c_z \xleftarrow{\$} \mathsf{DJ.Enc}_{N_D}(\boldsymbol{K}_z.\mathsf{fst})$.
4. Produce key $\boldsymbol{K}_z \leftarrow (\langle z \rangle_0, \langle \nu_{\ell_z} \cdot z \rangle_0)$ and garbled table (c_z, ct_x).

Multiplication – Evaluation

At evaluation time, given labels $\boldsymbol{L}_x$ and $\boldsymbol{L}_y$ as well as corresponding ciphertexts c_x, c_y, and ct_x, compute the label L_z as follows:

1. For $w \in \{x, y\}$, set $\langle w\rangle_1 \leftarrow \boldsymbol{L}_w.\mathsf{fst}$ and $\langle \nu_{\ell_w} \cdot w\rangle_1 \leftarrow \boldsymbol{L}_w.\mathsf{snd}$, then, for each $\ell \in [\ell_w, D-1]$, compute $\langle \nu_{\ell+1} \cdot w\rangle_1$ and $\langle d_\ell \cdot w\rangle_1$ as follows:
$$\langle d_\ell \cdot w\rangle_1 \leftarrow \langle w\rangle_1 - N_\ell^s \cdot \langle \nu_\ell \cdot w\rangle_1$$
$$\langle \nu_{\ell+1} \cdot w\rangle_1 \leftarrow \mathsf{DDLog}((c_\ell)^{\langle d_\ell \cdot w\rangle_1}) + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, (\mathsf{ord}(w), 1, \ell)) \bmod N^\zeta \ .$$
2. Compute sh_z as follows:
$$\begin{aligned}\langle z\rangle_1 \leftarrow &\big(\boldsymbol{L}_x.\mathsf{fst} \cdot \boldsymbol{L}_y.\mathsf{fst} - \mathsf{DDLog}((c_x)^{\langle d_D \cdot y\rangle_1}) - \mathsf{DDLog}((c_y)^{\langle d_D \cdot x\rangle_1}) \\ &+ \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, (\mathsf{ord}(w), 2, \ell)) \bmod N^\zeta\big) \ .\end{aligned}$$
3. Compute $\langle \nu_{\ell_z}^2 \cdot x\rangle_1 \leftarrow \mathsf{DDLog}((c'_{\ell_z-1})^{\langle d_{\ell_z-1} \cdot x\rangle_1}) + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, (\mathsf{ord}(w), 3)) \bmod N^\zeta$ then compute $(\langle d_\ell \cdot \nu_{\ell_z} \cdot x\rangle_1)_{\ell=\ell_z+1}^D$ recursively as:
$$\langle d_{\ell_z} \cdot \nu_{\ell_z} \cdot x\rangle_1 \leftarrow \langle \nu_{\ell_z} \cdot x\rangle_1 - N_\ell^s \cdot \langle \nu_{\ell_z}^2 \cdot x\rangle_1$$
For $\ell \in [\ell_z, D-1]$, $$\begin{aligned}\langle d_{\ell+1} \cdot \nu_{\ell_z} \cdot x\rangle_1 \leftarrow &\big(\langle \nu_{\ell_z} \cdot x\rangle_1 - N_\ell^s \cdot \mathsf{DDLog}((c'_\ell)^{\langle d_\ell \cdot \nu_{\ell_z} \cdot x\rangle_1}) \\ &+ \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, (\mathsf{ord}(w), 4, \ell)) \bmod N^\zeta\big) \ .\end{aligned}$$
Compute:
$$\begin{aligned}\langle \nu_{\ell_z} \cdot z\rangle_1 \leftarrow &\big(\langle \nu_{\ell_z} \cdot x\rangle_1 \cdot (\boldsymbol{L}_y.\mathsf{fst}) - \mathsf{DDLog}(c_y^{\langle d_D \cdot \nu_{\ell_z} \cdot x\rangle_1}) - \mathsf{DDLog}(\mathsf{ct}_x^{\langle d_{\ell_z-1} \cdot y\rangle_1}) \\ &+ \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, (\mathsf{ord}(w), 5)) \bmod N^\zeta\big) \ .\end{aligned}$$
4. Produce label $\boldsymbol{L}_z \leftarrow (\langle z\rangle_1, \langle \nu_{\ell_z} \cdot z\rangle_1)$.

Output: For each output value z, the garbler outputs:

– The garbled circuit, comprised of $(N_\ell)_{\ell\in[0,D]}$, $(c_\ell)_{\ell\in[0,D]}$, c_z for each input or multiplication z, ct_x for each x which is the left input of a multiplication, and $\mathsf{out}_z^G := \boldsymbol{K}_z.\mathsf{fst}$ for each output z.
– Its state (used to compute the input labels), comprised of $(N_\ell, \nu_\ell)_{\ell\in[0,D]}$, and $\boldsymbol{K}_x$ for each input x.

The evaluator outputs $(\mathsf{out}_z^E - \mathsf{out}_z^G)$ for each output z, where $\mathsf{out}_z^E := \boldsymbol{L}_z.\mathsf{fst}$.

Fig. 7: Arithmetic garbled circuit with two ciphertexts per multiplication (and free addition) from CPA-security of Damgård-Jurik (gate-by-gate description).

Theorem 11 (AGC with two ciphertexts per multiplication from CPA-secure Damgård-Jurik, via DDLog). *Let $\varepsilon > 0$. Let λ be a security parameter. Let $\zeta \geq 2$, assume the CPA-security of the Damgård-Jurik encryption scheme with parameter s, and let $\ell(\lambda)$ be the bit-length of the corresponding RSA modulus. For every $B \leq 2^{(\zeta-1-\varepsilon)\cdot\ell(\lambda)}$, the construction of Fig. 7 is an arithmetic garbling scheme for B-bounded integer computation.*

Moreover, if a circuit C has n inputs, m outputs, $s_\times$ multiplication gates, and has multiplicative depth D, then its garbled circuit has bit-size

$$(n+2s_\times+2D)\cdot\underbrace{(\zeta+1)\cdot\ell(\lambda)}_{\substack{\textit{size of a DJ}\\ \textit{ciphertext}}}+\underbrace{m\cdot\zeta\cdot\ell(\lambda)}_{\substack{\textit{decoding}\\ \textit{material}}}$$

while each input label is $2\zeta\cdot\ell(\lambda)$ bits.

Proof. The proof of Theorem 11 follows along the same lines as that of Theorem 9.

- ***Correctness.*** As in Theorem 9, it suffices to show that the following invariant is maintained:

  If w (at multiplicative depth ℓ_z) is either an input wire, or a wire whose two inputs x and y (at multiplicative depth ℓ_x and ℓ_y respectively) satisfy $\boldsymbol{L}_x=(1,\nu_{\ell_x})\cdot x+\boldsymbol{K}_x$ and $\boldsymbol{L}_y=(1,\nu_{\ell_y})\cdot y+\boldsymbol{K}_y$, then: with all but negligible probability (w.a.b.n.p.), $\boldsymbol{L}_z=(1,\nu_{\ell_z})\cdot y+\boldsymbol{K}_z$.

  We prove this by a case-by-case analysis.
  - If w is an input wire, this is true by definition of $\boldsymbol{L}_w$.
  - If $w=x\pm y$, assume the invariant holds for x and y.

    Lemma 12. *With all but negligible probabilty, $\langle\nu_{\ell_x+1}\cdot x\rangle_1-\langle\nu_{\ell_x+1}\cdot x\rangle_0=\nu_{\ell_x+1}\cdot x$.*

    Proof of Lemma 12. By correctness of the Damgård-Jurik cryptosystem, $\mathsf{DJ.Dec}_{N_{\ell_x},\zeta,\varphi_{\ell_x}}(c_\ell)=\nu_{\ell+1}$ w.a.b.n.p. By Lemma 7, $\mathsf{DDLog}(c_{\ell_x}^{\langle d_{\ell_x}\cdot x\rangle_1})-\mathsf{DDLog}(c_{\ell_x}^{\langle d_{\ell_x}\cdot x\rangle_0})=\nu_{\ell_x+1}\cdot x$ w.a.b.n.p. Analogously to Lemma 10, which amounts to a variant of Lemma 8 for shares which are only *pseudo*-random, $\langle\nu_{\ell_x+1}\cdot x\rangle_1-\langle\nu_{\ell_x+1}\cdot x\rangle_0=\nu_{\ell_x+1}\cdot x$ w.a.b.n.p. □

    We can inductively repeat the steps of the proof of Lemma 12 to show that, w.a.b.n.p., $\langle\nu_{\ell_w}\cdot x\rangle_1-\langle\nu_{\ell_w}\cdot x\rangle_0=\nu_{\ell_w}\cdot x$ and $\langle\nu_{\ell_w}\cdot y\rangle_1-\langle\nu_{\ell_w}\cdot y\rangle_0=\nu_{\ell_w}\cdot y$. Note that, since w is a multiplication gate, one of ℓ_x or ℓ_y is equal to ℓ_w. It follows that $\boldsymbol{L}_w-\boldsymbol{K}_w=(\boldsymbol{L}_x.\mathsf{fst}-\boldsymbol{K}_x.\mathsf{fst}+\boldsymbol{L}_y.\mathsf{fst}-\boldsymbol{K}_y.\mathsf{fst},\nu_{\ell_w}\cdot x+\nu_{\ell_w}\cdot y)=(1,\nu_{\ell_w})\cdot w$.
  - If $w=x\cdot y$, assume the invariant holds for x and y. Analogously to Lemma 12, $\langle w\rangle_1-\langle w\rangle_0=w$ w.a.b.n.p. Indeed, we can (1) show that $\mathsf{DDLog}((c_x)^{\langle d_D\cdot y\rangle_1})-\mathsf{DDLog}((c_x)^{\langle d_D\cdot y\rangle_0})\equiv y\cdot\langle x\rangle_0 \bmod N_{\ell_D}^\zeta$ and $\mathsf{DDLog}((c_y)^{\langle d_D\cdot x\rangle_1})-\mathsf{DDLog}((c_y)^{\langle d_D\cdot x\rangle_0})\equiv x\cdot\langle y\rangle_0 \bmod N_D^\zeta$, (2) use Eq. (3) to show that $\langle w\rangle_1-\langle w\rangle_0=x\cdot y\equiv w \bmod N_D^\zeta$ w.a.b.n.p., (3) apply the exact techniques of Lemma 10 to show that $\langle w\rangle_1-\langle w\rangle_0=w$. Analogously (and using Eq. (8)), $\langle\nu_{\ell_w}\cdot w\rangle_1-\langle\nu_{\ell_w}\cdot w\rangle_0=\nu_{\ell_w}\cdot w$ w.a.b.n.p. Therefore, $\boldsymbol{L}_w-\boldsymbol{K}_w=(1,\nu_{\ell_w}\cdot w)$ w.a.b.n.p.
- ***Privacy.*** Consider the following simulator $\mathcal{S}$ which, on input $(1^\lambda,C_\lambda,\boldsymbol{y}=(y_1,\ldots,y_m))$, does the following: (1) sample $k_{\mathsf{PRF}}\xleftarrow{\$}\{0,1\}^\lambda$ and $(N_\ell,\varphi_\ell)\xleftarrow{\$}\mathsf{DJ.KeyGen}(1^\lambda)$ for $\ell\in[D]$ then set $\nu_\ell\leftarrow N_\ell^{-\zeta} \bmod \varphi_\ell$ for $\ell\in[D]$, (2) sample $c_\ell,c'_\ell\xleftarrow{\$}\mathsf{DJ.Enc}_{N_\ell}(0)$ for $\ell\in[D]$, (3) for each input or multiplication w of C_λ, sample $c_w\xleftarrow{\$}\mathsf{DJ.Enc}_{N_D}(0)$, (4) for each wire x which is the left input of a multiplication gate w (at multiplicative depth ℓ_w), sample $\mathsf{ct}_{N_{\ell_w-1}}(0)$, (5) if $w_1,\ldots,w_m$ are the outputs of C_λ, sample $\mathsf{out}_w^E\xleftarrow{\$}(-N_D^\zeta/2,N_D^\zeta/2]$ then set $\mathsf{out}_{w_i}^G\leftarrow\mathsf{out}_{w_i}^E-y_i$, (6) for each input x of C_λ, sample $\boldsymbol{L}_x\xleftarrow{\$}[N_1^\zeta]$, (7) set $\widehat{C}_\lambda\leftarrow((N_\ell)_{\ell\in[D]},(c_\ell)_{\ell\in[D]},(c'_\ell)_{\ell\in[D]},(c_w)_w,(\mathsf{ct}_w)_w,(\mathsf{out}_w^G)_w)$, and output $((\boldsymbol{L}_x)_x,\widehat{C}_\lambda)$.

  Let us show, via a hybrid argument, that this simulator satisfies the privacy requirement of Definition 1.
  1. Starting from the real-world experiment, for each output wire w_i we replace $\mathsf{out}_{w_i}^G$ with $\mathsf{out}_{w_i}^E-y_i$. These two hybrids are statistically indistinguishable by correctness of the garbling scheme (established earlier in this proof).

2. For $\ell = 1$ to D (in this order), we can switch out all encryptions performed under key N_ℓ with encryptions of 0, by CPA security of the Damgård-Jurik cryptosystem. Once all these ciphertexts have been switched out, the resulting hybrid coincides with the ideal world.

– ***Size.*** The garbled circuit is comprised of $(n + 2s_\times + 2D)$ Damgård-Jurik ciphertexts (c_x for each input x, c_z for each multiplication z, ct_x for each wire x which is the left input of a multiplication gate, and c_ℓ and c'_ℓ for each of the D multiplicative layers of the circuit), as well as the garbler's share of each output (out^G_z for each output z).

□

4.3 Optimised left-right wire assignment.

In Sections 4.1 and 4.2, we presented two arithmetic garbling scheme which we now refer to as "the KDM scheme" and "the CPA scheme", respectively. Recall that, for each multiplication gate $z = x \cdot y$, the garbled circuit for the KDM scheme should contain (the means to compute) encryptions of the form $\mathsf{Enc}_N(\langle \varphi \cdot x \rangle_0)$ and $\mathsf{Enc}_N(\langle \varphi \cdot y \rangle_0)$, while the garbled circuit for the CPA scheme should contain (the means to compute) encryptions of the form $\mathsf{Enc}_{N_D}(\langle x \rangle_0)$, $\mathsf{Enc}_{N_D}(\langle y \rangle_0)$, and $\mathsf{Enc}_{N_{\ell_z - 1}}(\langle \nu_{\ell_z} \cdot x \rangle_0)$.

On the face of it, the schemes should achieve rate 1/2 and 1/3 respectively. However, we already presented the following optimisation based on linear homomorphism: if the garbled circuit contains a ciphertext $\mathsf{Enc}_N(\langle \varphi \cdot w \rangle_0)$ (for the KDM scheme) or $\mathsf{Enc}_{N_D}(\langle w \rangle_0)$ (for the CPA scheme) for each w which is either an input or the result of a multiplication gate, then the parties can in fact compute such ciphertexts for every single wire of the circuit—and in particular for every input of a multiplication gate, which is what they need—(using linear homomorphism to deal with addition gates). The garbled circuits now contain one (KDM) or two (CPA) ciphertext per multiplication and one ciphertext per input, and the garbling schemes therefore have rate 1 (KDM) and 1/2 (CPA).

In the CPA scheme, we only need to give out a ciphertext of the form $\mathsf{Enc}_{N_{\ell_z - 1}}(\langle \nu_{\ell_z} \cdot x \rangle_0)$ *for the left input wire* of multiplication z. Because multiplications are commutative, we have some degree of liberty in deciding which wires are left ones. If a value is used as the input to many multiplication gates at the same multiplicative depth, it may be beneficial to make this value the left input to these multiplications, in order to reuse the same ciphertext.

4.3.1 Reducing left-right assignments to vertex cover. In this section, we show how the problem of choosing which wires should be left or right inputs to multiplication gates in order to minimise the size of the garbled circuit can be reduced to the problem of finding (small) vertex covers in some graphs.

Definition 13 (Depth-i multiplication graph of a circuit). *Let C be a fan-in two arithmetic circuit of multiplicative depth D. For each $i \in [D]$, we define the* depth-i multiplication graph of C *as the graph $G_i = (V, E_i)$ where V is the set of all inputs and computation gates of C, and E_i is defined by: $(x, y) \in E_i$ if and only if there exists a gate z at multiplicative depth i such that $z = x \cdot y$.*

Lemma 14 (Left-right wire assignment based on vertex covers). *Let C be a fan-in two arithmetic circuit of multiplicative depth D. For $i \in [D]$, let G_i be the depth-i multiplication graph of C, and F_i be a vertex cover of G_i.*
*Up to permuting the input wires of multiplication gates (*e.g. *computing $z = x \cdot y$ as $z = y \cdot x$), we can assume that, for every $i \in [D]$, all left input wires to a depth-i multiplication gate are in F_i.*

Proof. Let $i \in [D]$, let z be a depth-i multiplication gate of C, and let x, y be the inputs to z. By definition of G_i, $(x, y) \in E_i$. By definition of a vertex cover, at least one of (the nodes of G_i associated with) x or y must be in F_i: we can therefore always ensure the left input wire is in F_i. □

It follows from Lemma 14 and Section 4.2.2 that if we can establish a bound of the form $(\sum_{i=1}^{D} |F_i|/|C|) \leq c$ for every circuit C (from a given class), then assuming the IND-CPA security of the Damgård-Jurik encryption scheme, there is an arithmetic garbling scheme (for that class) with rate $1/(1+c)$.[9]

4.3.2 A constructive upper bound for levelled circuits. In this section we show that the CPA scheme can be adapted to achieve rate 3/5 for levelled circuits[10], but at the cost of losing free addition. While losing free addition makes the scheme concretely better only for circuits with at least $83.\bar{3}\%$ of multiplication gates, this tradeoff is still interesting in a theoretical sense because rate is defined as a worst-case quantity over all circuits (in particular those containing only multiplication gates).

Lemma 15 ("Caro-Wei theorem", Bounding the independence number of a graph in terms of its degree sequence, [Car79, Wei81]). *Let $G = (V, E)$ be a simple graph. Its independence number $\alpha(G)$ satisfies:*

$$\alpha(G) \geq \sum_{u \in V} \frac{1}{\deg(u) + 1} .$$

The "Caro-Wei theorem" was originally proven via a probabilistic argument, but a constructive proof was later provided by Murphy [Mur91]. For completeness, we recall this construction in Fig. 8.

Algorithm IndSet

```
On input a graph $G = (V, E)$:

1. If $V = \varnothing$, return $\varnothing$.
2. Else, choose $u \in V$ of minimum degree in $G$.
3. Output $\{u\} \cup \mathsf{IndSet}(G[V \setminus N_G[u]])$.
```

Fig. 8: Murphy's [Mur91] greedy recursive algorithm to find an independent set satisfying the Caro-Wei bound [Car79, Wei81].

Lemma 16 (Two-thirds bound for levelled circuits). *Let C be an n-input, m-output, multiplicative-depth-D levelled circuit. For $i \in [D]$, let G_i be the depth-i multiplication graph of C (Definition 13).*
There exists a polynomial-time algorithm for finding $F_1, \dots, F_D$ such that: (1) for each $i \in [D]$, F_i is a vertex cover of G_i, and (2) $(\sum_{i=1}^{D} |F_i|/|C|) \leq 2/3$.

Proof. Define the following quantities:

[9] Note that if we can provide a bound of the form $(\sum_{i=1}^{D} |F_i|/s_\times(C)) \leq c$, where $s_\times(C)$ is the number of multiplication gates of circuit C, then the scheme has free addition.

[10] A circuit is *levelled* if its nodes (inputs and computation gates) can be partitioned into layers such that each edge connects nodes in adjacent layers. The difference between a levelled and a *layered* circuit is that, in the latetr, any input to the circuit is allowed to be an input to gates in different layers.

– For each $i \in [D]$, G_i is the depth-i multiplication graph of C;
– For each $i \in [D]$ and for $j \in [0, |V|]$, $n_{i,j} \coloneqq |\{u \in V \colon \deg_{G_i}(u) = j\}|$;
– For each $i \in [0, D]$, $s_{\times,i}$ (resp. s_i) is the size of (resp. the number of multiplication gates in) the i^{th} layer of C.

Since the number of edges in a graph is twice the sum of the degrees of its nodes,

$$\forall i \in [D], 2 \cdot s_i \geq 2 \cdot s_{\times,i} = 2 \cdot |E_i| = \sum_{u \in V} \deg_{G_i}(u) = \sum_{j=0}^{|V|} j \cdot n_{i,j} \ . \tag{9}$$

The complement of an independent set is a vertex cover [Gal59], so, by Lemma 15, G_i admits a vertex cover F_i of size at most $|V| - \sum_{u \in V} \frac{1}{\deg_{G_i}(u)+1}$.

$$\forall i \in [D], |F_i| = \sum_{u \in V} \left(1 - \frac{1}{\deg_{G_i}(u) + 1}\right) = \sum_{u \in V} \frac{\deg_{G_i}(u)}{\deg_{G_i}(u) + 1} = \sum_{j=0}^{|V|} n_{i,j} \cdot \frac{j}{j+1} \ . \tag{10}$$

$$\forall i \in [D], s_{i-1} \geq \sum_{j=0}^{|V|} n_{i,j} \ . \tag{11}$$

By considering the linear combination "$\sum_{i=1}^{D}(\frac{1}{6}[Eq.\ (9)] + \frac{1}{3}[Eq.\ (11)] - [Eq.\ (10)])$" we obtain Eq. (12):

$$\begin{aligned} \tfrac{s_D}{3} + \tfrac{2s}{3} + \tfrac{s_0}{3} - \sum_{i=1}^{D} |F_i| = \sum_{i=1}^{D} \left(\tfrac{2s_i}{6} + \tfrac{s_{i-1}}{3} - |F_i|\right) &\geq \sum_{i=1}^{D} \left(\sum_{j=0}^{|V|} \left(\tfrac{j}{6} + \tfrac{1}{3} - \tfrac{j}{j+1}\right) \cdot n_{i,j}\right) \\ &= \sum_{i=1}^{D} \sum_{j=0}^{|V|} \left(\tfrac{(j-2)(j-1)}{6(j+1)} \cdot n_{i,j}\right) \\ &= \sum_{i=1}^{D} \sum_{j=3}^{|V|} \left(\tfrac{(j-2)(j-1)}{6(j+1)} \cdot n_{i,j}\right) \geq 0 \ . \end{aligned} \tag{12}$$

Because all gates in the final layer must be outputs, $s_D \leq m$. Because all gates in the first layer must be inputs, $s_0 \leq n$. Combining these facts with Eq. (12), $\frac{n+m+2s}{3} \geq \sum_{i=1}^{D} |F_i|$. □

Note that the bound of Lemma 16 is tight in the sense there exist circuits for which any vertex covers $(F_i)_{i \in [D]}$ must satisfy $\sum_i = 1^D |F_i| \geq 2|C|/3$ (consider for instance a circuit such that each G_i is a union of node-disjoint triangles).

Lemma 16 yields an arithmetic garbling scheme with rate $1/(1 + 2/3) = 3/5$ for levelled circuits.

Acknowledgments

This research was supported by: the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement number 803096 (SPEC); the Danish Independent Research Council under Grant-IDs DFF-3103-00077B (CryptoDigi) and DFF-0165-00107B (C3PO); and the DARPA SIEVE program (contract HR001120C0085 "FROMAGER"). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited).

References

AIK11. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 120–129. IEEE Computer Society Press, October 2011.

ARS24. Damiano Abram, Lawrence Roy, and Peter Scholl. Succinct homomorphic secret sharing. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024*, pages 301–330, Cham, 2024. Springer Nature Switzerland.

BGG$^+$14. Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556. Springer, Heidelberg, May 2014.

BGI16. Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 509–539. Springer, Heidelberg, August 2016.

BLLL23. Marshall Ball, Hanjun Li, Huijia Lin, and Tianren Liu. New ways to garble arithmetic circuits. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 3–34. Springer, Heidelberg, April 2023.

BMR16. Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for Boolean and arithmetic circuits. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 565–577. ACM Press, October 2016.

BRS03. John Black, Phillip Rogaway, and Thomas Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002*, volume 2595 of *LNCS*, pages 62–75. Springer, Heidelberg, August 2003.

Car79. Yair Caro. New results on the independence number. Technical report, Technical Report, Tel-Aviv University, 1979.

DJ01. Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In Kwangjo Kim, editor, *PKC 2001*, volume 1992 of *LNCS*, pages 119–136. Springer, Heidelberg, February 2001.

FMS19. Nils Fleischhacker, Giulio Malavolta, and Dominique Schröder. Arithmetic garbling from bilinear maps. In Kazue Sako, Steve Schneider, and Peter Y. A. Ryan, editors, *ESORICS 2019, Part II*, volume 11736 of *LNCS*, pages 172–192. Springer, Heidelberg, September 2019.

Gal59. T Gallai. Clber extreme punkt und kantenmengen. *Ann. Univ. Sci. Budapest, E6tv6s Sect. Math*, 2, 1959.

GKP$^+$13. Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 555–564. ACM Press, June 2013.

Hea24. David Heath. Efficient arithmetic in garbled circuits. 2024. `https://eprint.iacr.org/2024/139`.

HVDH21. David Harvey and Joris Van Der Hoeven. Integer multiplication in time o(nlog\,n). *Annals of Mathematics*, 193(2):563–617, 2021.

HY24. Carmit Hazay and Yibin Yang. Toward malicious constant-rate 2pc via arithmetic garbling. 2024. `https://eprint.iacr.org/2024/283`.

LL24. Hanjun Li and Tianren Liu. How to garble mixed circuits that combine boolean and arithmetic computations. 2024. `https://eprint.iacr.org/2023/1584`.

Mur91. Owen Murphy. Lower bounds on the stability number of graphs computed in terms of degrees. *Discrete Mathematics*, 90(2):207–211, 1991.

OSY21. Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 678–708. Springer, Heidelberg, October 2021.

Pai99. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999.

RS21. Lawrence Roy and Jaspal Singh. Large message homomorphic secret sharing from DCR and applications. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 687–717, Virtual Event, August 2021. Springer, Heidelberg.

Wei81. Victor K Wei. A lower bound on the stability number of a simple graph, 1981.

Yao82. Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.