A NOTE ON THE SECURITY OF THE BITVM3 GARBLING SCHEME

A. FUTORANSKY, G. LAROTONDA, AND F. BARBARA

Abstract. We provide minimal counterexamples for the security of the BitVM3 garbling scheme presented in [Lin25]: our attack allows the evaluator to forge input and output wires. Then we use the same idea to exhibit an attack on the forward label propagation garbling scheme proposed in [FDZ25]. In both cases, the *authenticity* property of the garbling scheme is broken.

Keywords: authenticity, BitVM3, garbling scheme, RSA

1. RSA garbling

In the BitVM3 paper [Lin25] a garbling scheme based on RSA encryption is introduced, where the *authenticity* of the scheme is assumed to follow from the hidden order of the group. In this short note we show that this assumption is wrong and we provide minimal counterexamples[1].

Remark 1 (Coprimes and the extended Euclidean algorithm)**.** The BitVM3 scheme is based on modular arithmetic in $\mathbb{Z}_N$: the garbler P (he) chooses $N = PQ = (2p+1)(2q+1)$ with secret safe primes p, q. Recovering x from x^a is unfeasible for the evaluator E (she), who doesn't know $\varphi(N) = 4pq$ and cannot compute exponent inverses. The key idea of our attack is as follows: recovering x from x^a or from x^b is unfeasible for the evaluator. But when a, b are coprime it is feasible for the evaluator to find integers k_1, k_2 such that $ak_1 + bk_2 = 1$ by means of the extended Euclidean algorithm, which does not require knowledge of $\varphi(N)$. Then knowing x^a, x^b allows her to learn x simply by calculating

$$x = x^{k_1 a + k_2 b} = (x^a)^{k_1}(x^b)^{k_2}.$$

1.1. **BitVM3 notation and rules.** In the BitVM3 scheme, the wires are computed using 5 public exponents e, e_1, e_2, e_3, e_4, invertible modulo $\varphi(N)$; small primes (3,5,7,11,13) are suggested. Denoting d_i their inverses, it is also assumed that $e_1 e_4 d_2 - e_3$ is invertible $\pmod{pq}$. Following the notation in [Lin25], let us indicate wires with

(A. Futoransky, F. Barbara. `futo@fairgate.io, fadi.barbara@fairgate.io`) Fairgate Labs.
(G. Larotonda. `glaroton@dm.uba.ar`) FCEyN-UBA & CONICET.

[1]While finishing this note and the PoC, we learned that L. Eagen published a somewhat related attack to the BitVM3 garbling scheme in https://hackmd.io/@liameagen/bitvm3-garbling-note

small letters a, b, c, d etc. We use a_0 to indicate a label in $\mathbb{Z}_N$ carrying the bit 0 in wire a, and we use a_1 to indicate the label of the same wire carrying the bit 1. The rules for an AND gate are as follows: here a, b are the input wires and x is the output wire.

$$\begin{aligned} x_0 &= a_0^e \cdot b_0^{e_1} &\pmod N \\ x_0 &= a_0^e \cdot b_1^{e_2} &\pmod N \\ x_0 &= a_1^e \cdot b_0^{e_3} &\pmod N \\ x_1 &= a_1^e \cdot b_1^{e_4} &\pmod N. \end{aligned} \tag{1}$$

In what follows, let us agree to call a in the table above the *left* input wire (all powers equal to e) and b the *right* input wire.

From this rules, knowing the labels x_0, x_1, the prover P who knows $\varphi(N)$, is able to compute a_0, a_1, b_0, b_1 to move backward in the garbled circuit label definition. But if two gates are fed from the same input wire b (fan-out>1), since the y_0, y_1 of the second gate are different, this leads to different values b'_0, b'_1 which is inconsistent. See Figure 1 below: b_0, b_1 produced from gate X (backwards from x_0, x_1 and equations) will be different to b'_0, b'_1 produced from gate Y (backwards from y_0, y_1).
To remedy this, the solution proposed in BitVM3 is to make the garbler compute *adaptors* T as follows

$$T_{b',0} = \frac{b'_0}{b_0}, \qquad T_{b',1} = \frac{b'_1}{b_1} \pmod N$$

and make the prover share also these adaptors $T_{b',i}$ to the evaluator as circuit material on the setup. Please note that the adaptors are invertible modulo N, so knowing b_0 is equivalent for the evaluator to know b'_0, and likewise with b_1.
Then, the evaluator, obtaining for instance the label b_0 for wire b upon her evaluation, uses it (and the label from wire a) to produce the output of gate X. But instead of using it to evaluate Y, she first multiplies by the corresponding adaptor, to obtain

$$b'_0 := b_0 \cdot T_{b',0} \pmod N$$

and uses this nonce to evaluate Y (and the one from wire c). In case she has b_1, she uses the other adaptor to obtain b'_1.

2. The attack on BitVM3

Now we get into the details of the implementation of our attack. We will construct a minimal 2-gate, 3-inputs circuit that forces the garbler into reusing one of the input wires. This results in the use of adaptors. We will exploit this and the fact that combinations of RSA encryptions (induced by the adaptors) makes possible to

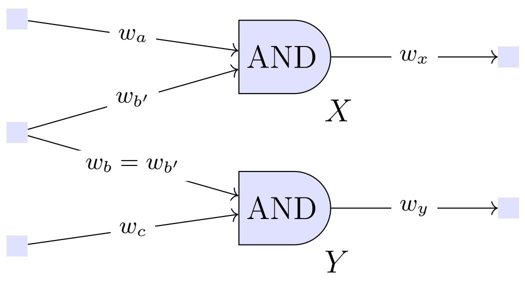

FIGURE 1. Minimal circuit with two parallel AND gates reusing an input wire $w_b = w_{b'}$.

recover messages. This will enable the evaluator to forge one input and one output wire of the circuit.

Assume the circuit is as in Figure 1. We set the rules for X and Y as follows (we've chosen the reused wire to be on the right in gate X and to be on the left on gate Y):

$$\begin{aligned}
x_0 &= a_0^e \cdot (b_0')^{e_1} \pmod N & y_0 &= b_0^e \cdot c_0^{e_1} \pmod N\\
x_0 &= a_0^e \cdot (b_1')^{e_2} \pmod N & y_0 &= b_0^e \cdot c_1^{e_2} \pmod N\\
x_0 &= a_1^e \cdot (b_0')^{e_3} \pmod N & y_0 &= b_1^e \cdot c_0^{e_3} \pmod N\\
x_1 &= a_1^e \cdot (b_1')^{e_4} \pmod N. & y_1 &= b_1^e \cdot c_1^{e_4} \pmod N.
\end{aligned}$$

In this setting, we assume at this stage the evaluator has labels a_0, b_0, c_0, and learns b_0' using the adapter. Evaluating gate X, she gets x_0. From gate Y she gets y_0, the legitimate output corresponding to both bits equal to 0. Now the evaluator is in possession of

$$T_{b',0},\ T_{b',1},\ a_0, b_0, c_0, b_0'.$$

Lemma 2. *In the circuit of Figure 1, an evaluator knowing input labels a_0, b_0, c_0 is able to forge input label b_1.*

Proof. From the first gate X rules

$$a_0^e \cdot (b_0')^{e_1} = x_0 = a_0^e \cdot (b_1')^{e_2} \pmod N,$$

the evaluator learns $(b_1')^{e_2} = (b_0')^{e_1} \pmod N$, and by means of the adaptor, she learns $b_1^{e_2}$. From gate Y rules

$$b_1^e \cdot c_0^{e_3} = y_0 = b_0^e \cdot c_0^{e_1} \pmod N,$$

she learns

$$b_1^e = b_0^e \cdot c_0^{e_1 - e_3} \pmod N.$$

She now looks up for k_1, k_2 such that $k_1 e + k_2 e_2 = 1$, and from this she learns

$$b_1 = (b_1^e)^{k_1} \cdot (b_1^{e_2})^{k_2} \pmod N.$$ □

Corollary 3. *In the same setting as the previous lemma, the evaluator is able to forge the output label x_1.*

Proof. From the last two rules for gate X it follows that

$$x_1 = x_0 \cdot (b_1')^{e_4} \cdot (b_0')^{-e_3} \pmod N. \qquad \square$$

Remark 4. Most non-trivial circuits will suffer from many similar exponent collisions. Trying to transform the topology to avoid this type of attack is not possible.

3. The attack on the forward label propagation scheme

In this scheme proposed in [FDZ25], the rules relating input/outputs of a gate are the same as in (1) but now the garbler P first chooses the input labels of the circuit and propagates them forward according to these rules, using *output* adaptors specific for each type of gate. We provide a minimal example breaking security, following the same ideas that in the previous section, and refer the reader to the mentioned paper for details: we construct a similar attack, specific to this garbling scheme.

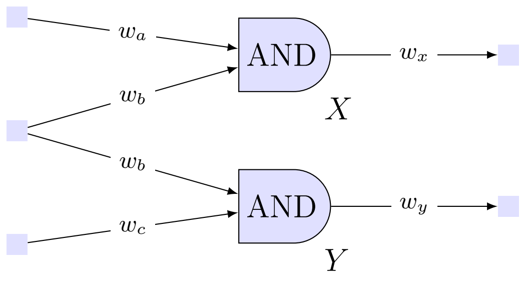

FIGURE 2. Minimal circuit with two parallel AND gates reusing input wire w_b.

$$\begin{aligned} x_0 &= a_0^e \cdot b_0^{e_1} \pmod N & y_0 &= b_0^e \cdot c_0^{e_1} \pmod N \\ x_1 &= a_0^e \cdot b_1^{e_2} \pmod N & y_1 &= b_0^e \cdot c_1^{e_2} \pmod N \\ x_2 &= a_1^e \cdot b_0^{e_3} \pmod N & y_2 &= b_1^e \cdot c_0^{e_3} \pmod N \\ x_3 &= a_1^e \cdot b_1^{e_4} \pmod N. & y_3 &= b_1^e \cdot c_1^{e_4} \pmod N. \end{aligned}$$

These are the rules for gates X, Y, and what follows are the adaptors for the respective AND gate outputs (Table 3 in [FDZ25]):

$$A_1 = \frac{x_0}{x_1}, A_2 = \frac{x_0}{x_2}, \qquad A_1' = \frac{y_0}{y_1}, A_2' = \frac{y_0}{y_2}.$$

These are passed from garbler to evaluator as circuit material. Assume that the evaluator gets labels a_0, b_0, c_0. The legitimate output of our minimal example is then x_0, y_0.

From the rules of gate X, it follows that

$$a_0^e \cdot b_0^{e_1} = x_0 = A_1 \cdot x_1 = A_1 \cdot a_0^e \cdot b_1^{e_2}.$$

Hence our evaluator gets her hands on $b_1^{e_2} = A_1^{-1} \cdot b_0^{e_1}$. From the rules of gate Y, it follows that

$$b_0^e \cdot c_0^{e_1} = y_0 = A_2' y_2 = A_2' b_1^e \cdot c_0^{e_3},$$

and our evaluator now has

$$b_1^e = (A_2')^{-1} \cdot b_0^e \cdot c_0^{e_1 - e_3}.$$

Repeating the argument of Lemma 2, we have proved the following:

Corollary 5. *An evaluator holding input labels a_0, b_0, c_0 of circuit in Figure 2 is able to forge input label b_1.*

4. Regarding linear adaptors in the original BitVM3 scheme

Now we explore an alternative adaptor method suggested by Robin Linus, where each label's adaptor is an affine function:

$$b_0' = b_0 T_{0,1} + T_{0,2}, \qquad b_1' = b_1 T_{1,1} + T_{1,2} \pmod N.$$

In this case, we construct a new two ANDs example, with the reused wire on the left input of both gates. Resulting in the following equations:

$$\begin{aligned}
x_0 &= (b_0')^e \cdot a_0^{e_1} \pmod N & y_0 &= b_0^e \cdot c_0^{e_1} \pmod N\\
x_0 &= (b_0')^e \cdot a_1^{e_2} \pmod N & y_0 &= b_0^e \cdot c_1^{e_2} \pmod N\\
x_0 &= (b_1')^e \cdot a_0^{e_3} \pmod N & y_0 &= b_1^e \cdot c_0^{e_3} \pmod N\\
x_1 &= (b_1')^e \cdot a_1^{e_4} \pmod N. & y_1 &= b_1^e \cdot c_1^{e_4} \pmod N.
\end{aligned}$$

The known labels for the evaluator are again a_0, b_0, c_0, with legitimate outputs x_0, y_0.

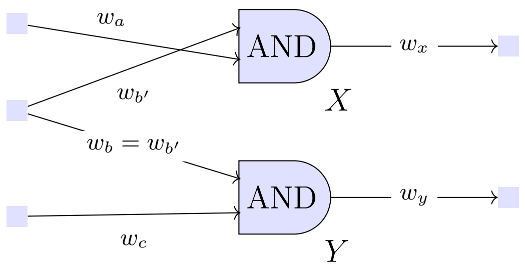

FIGURE 3. Two AND gates reusing an input wire $w_b = w_{b'}$.

Theorem 6. *Given the input labels a_0, b_0, c_0 in the circuit of Figure 3, the evaluator can forge the input label b_1.*

Proof. From the rules of gate X the evaluator gets $(b_1')^e = x_0 \cdot a_0^{-e_3}$, and likewise she gets b_1^e from the rules of gate Y. Using the adaptors she knows

$$(b_1')^e = (b_1 T_{1,1} + T_{1,2})^e \pmod N$$

So simply put, the task is to recover $b_1 \pmod N$ from known h_1, h_2 where

$$h_1 = b_1^e \qquad h_2 = (\alpha b_1 + \beta)^e$$

where $\alpha = T_{1,1}, \beta = T_{1,2}, e$ are known to the evaluator. Name

$$z = \alpha\beta^{-1} b_1, \quad c_1 = \alpha^e \beta^{-e} h_1, \quad c_2 = \beta^{-e} h_2.$$

Note that the previous problem is equivalent to recover z from

$$c_1 = z^e \qquad \text{and} \qquad c_2 = (z+1)^e \pmod N.$$

This can be done following the guidelines of [CFPR89]: the polynomials $a(t) = t^e - c_1$ and $b(t) = (t+1)^e - c_2$ have the same degree and a common root $t = z$. By means of the Euclidean algorithm for polynomials she can find polynomials $f, h \in \mathbb{Z}_N[t]$ such that

$$g(t) = f(t)(t^e - c_1) + h(t)((t+1)^e - c_2)), \tag{2}$$

where g is the g.c.d. of our polyomials, and must be a monic polynomial. It is clear that $t - z$ divides $g(t)$, but in fact $g(t) = t - z$ (except possibly for a finite number of values of z, see Remark 7 below). Evaluating (2) in $t = 0$ she gets

$$-z = 0 - z = -f(0) \cdot c_1 + h(0) \cdot (1 - c_2) \pmod N,$$

recovering z and learning b_1. □

Remark 7. Please note that although there might a finite number of possible exceptions corresponding to other common roots in some extension field of $\mathbb{Z}_P$, as suggested in the footnote of page 2 of [CFPR89], this algorithm returns a linear polynomial with very high probability.

References

[CFPR89] Don Coppersmith, Matthew Franklin, Jacques Patarin, Michael Reitert. *Low-Exponent RSA with Related Messages.* Proc. of Eurocrypt'96, LNCS 1070, pp. 1-9. https://link.springer.com/content/pdf/10.1007/3-540-68339-9_1.pdf

[FDZ25] Alva Fu, Stephen Duan, Ethan Zhu. *Instantiating BitVM3 from Label Forward Propagation*, https://www.goat.network/bitvm3-label-forward-propagation

[Lin25] Robin Linus. *BitVM3: Efficient Computation on Bitcoin*, https://bitvm.org/bitvm3.pdf