THC: trustable homomorphic computation

Pablo Rauzy
Université Paris 8 (Saint-Denis, France)
https://pablo.rauzy.name/ — pr@up8.edu

Abstract

Homomorphic cryptography is used when computations are delegated to a third party which is not trusted with the data, often for privacy concerns. However, the same third party is trusted with the program, i.e., it is assumed that it will not tamper with the desired sequence of operations to execute on the encrypted data. In cases when homomorphic cryptography is used to outsource resources-greedy computations on personal data, for example from a smartphone to the cloud, trusting the third party with the program is quite bothersome. We aim at providing a simple and cost-effective way around this problem. By leveraging methods developed to protect embedded asymmetric cryptography computations against fault injection attacks, we could remove the necessity to trust the third party with the program, and thus the need to trust the third party at all. Our goals include the development of a library to ease and automate the use of the proposed method.

Context and Related Work

Delegating computation to a third party is very common nowadays, with the proliferation of small devices like smartphones and tablets which are mostly terminal interfaces for cloud services. This tendency is even accelerating. Indeed, a lot of low-power and low-performance devices are now getting connected since the internet of things is becoming a reality. Most of these devices are made to make people's life better, but part of the process is the monitoring of personal user data, for example smartwatches may collect the position and the level of physical activities of their wearer.

Hence, serious privacy concerns need to be addressed. When data only need to be stored or transmitted from user to user, classical cryptography (symmetric and asymmetric) can solve the problem. However, most of the time users' data have to be processed, e.g., to generate statistics or to compute quantities that are more informational than the raw values collected by the devices. Homomorphic cryptography allows users to encrypt their data before they are sent to the cloud for further processing. Computations can then be performed on the encrypted values, and the result can be sent back to the user who is the only one able to decrypt it.

While this may be enough to solve the privacy issue (depending on the definition of privacy), it is not enough for users to be able to fully trust the third party performing the computation on their homomorphically encrypted data. Indeed, there is no reason to trust the third party with the execution of the expected sequence of operations.

For the sake of simplicity, imagine the following dummy example:
An insurance company offers multiple options (at different prices) to their clients depending on how well they want to be covered for weight-related diseases. In order to help their clients to choose the option that better suit their need, the insurance company offers a service that watches their body mass index (BMI) over time. $BMI = \frac{\text{mass}}{\text{height}^2}$.

However, people do not want their private data such as mass and height to be sent in clear over the network, nor to be revealed to their insurance company. This is where homomorphic cryptography can help. Here, users' devices would send m' and h' the homomorphically encrypted values of the users' mass and height to the insurance's cloud service, and classically users would have to trust the insurance to compute their BMI correctly, and not something like $\frac{\text{mass}+20}{\text{height}^2}$ which would make the users think they are overweight and should subscribe to the corresponding, more expensive option.

What we would like to have is *trustable homomorphic computations*, i.e., to allow users' devices to verify that the cloud service has performed the expected sequence of operations on the transmitted values. And of course we want to be able to do that with the constrained resources of low-resources devices. In our example it does not really make sense because the BMI computation can always be done locally, however low the performances of the device are, but in real-world situations dealing with e.g., complex health or position data, it is important for the verification to be cost-effective.

There are existing works on the subject [8, 7, 5, 6]. However, these attempts at verifiable delegation of homomorphic computation are either impractical and/or introduce complex cryptographic constructions and rely on them. For example, the work of Lai et al. [6], which is the closest to what we want to achieve ourselves, introduce a new cryptographic primitive called "homomorphic encrypted authenticator", and stay at a theoretical level (it does not provide an implementation). We see that as a problem for several reasons: their complexity hinders their adoption in real products, there is no reference implementation provided that one could use to implement the scheme in real products, they may be too costly for small devices, and in addition, these new cryptographic constructions may be broken in the future.

The Proposed Approach

Instead of relying on new and complex cryptographic constructions, we propose to adapt a tried and tested method from another area of cryptology, namely implementation security against physical attacks.

Indeed, countermeasures against fault injection attacks do just what we want: they verify the integrity of the computation to make sure that it has not been tampered with. Although the setting is quite different, it is possible to capture the tampering in both cases using the same formal model (e.g., modification of intermediate values, instruction change, etc.), and thus to take advantage of the work that has already been done in the field of fault injection attacks.

One of these countermeasures which have been extensively used and studied is called *modular extension*, and consists in lifting the computation into an over-structure (e.g., an overring $\mathbb{Z}_{pr}$) which allows to quotient the result of the computation back to the original structure (e.g., $\mathbb{F}_p$), as well as quotienting a "checksum" of the computation to a smaller structure (e.g., $\mathbb{F}_r$). What has just been described is the *direct product* of the underlying algebraic structures. If an equivalent computation is performed in parallel in the smaller structure, its result can be compared with the checksum of the main computation. If they are the same, we have a high confidence in the integrity of the main computation. This protection is sketched in Fig. 1.

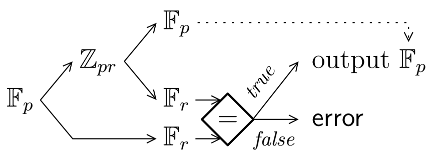

Figure 1: Sketch of the principle of *modular extension.*

A nice property of the modular extension scheme is that the small computations over $\mathbb{F}_r$ is entirely independent from the combined one over $\mathbb{Z}_{pr}$, and can thus be performed on another machine. Hence the applicability of modular extension to the verification of the integrity of delegated homomorphic computations.

Modular extension has been successfully ported from CRT-RSA, for which it has originally been designed by Shamir in 1999 [11], to elliptic curve scalar multiplication [2, 1], and has been formally studied in both settings [9, 10, 4]: the non-detection probability is provably inversely proportional to the security parameter r (the size of the small structure).

Although there exists homomorphic schemes that rely on modular arithmetic (such as YASHE [3]), it is still an open question whether some of them are well suited for modular extension, i.e., if they have no tests depending on the modular values. If no such scheme exists, the modular extension scheme could be extended to cover tests on modular values, for example using Lagrangian interpolation polynomial, which will have to be optimized. Another solution would be to devise a new variant of homomorphic computation scheme that plays well with modular extension.

After that, the new trustable homomorphic computation scheme will be formally studied, in particular security-wise with the theoretical probability of not detecting that the homomorphic computation has been tampered.

Once the theoretical work has been done, modular extension is sufficiently simple in principle to be implemented efficiently in practice, so we will provide a reference library implementing verifiable delegation of homomorphic computations. This implementation will be used to make performance measurements, and to verify that the practical probability of non-detection corresponds to what theory predicted.

References

[1] Y.-J. Baek and I. Vasyltsov. How to Prevent DPA and Fault Attack in a Unified Way for ECC Scalar Multiplication - Ring Extension Method. In *Information Security Practice and Experience.* 2007. (I can provide the PDF if necessary).

[2] J. Blömer, M. Otto, and J.-P. Seifert. Sign Change Fault Attacks on Elliptic Curve Cryptosystems. In *Fault Diagnosis and Tolerance in Cryptography.* 2006. `https://eprint.iacr.org/2004/227`.

[3] J. W. Bos, K. Lauter, J. Loftus, and M. Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In *Cryptography and Coding*. 2013. https://eprint.iacr.org/2013/075.

[4] M. Dugardin, S. Guilley, M. Moreau, Z. Najm, and P. Rauzy. Using modular extension to provably protect edwards curves against fault attacks. Cryptology ePrint Archive, Report 2015/882, 2015. https://eprint.iacr.org/2015/882.

[5] D. Fiore, R. Gennaro, and V. Pastro. Efficiently verifiable computation on encrypted data. Cryptology ePrint Archive, Report 2014/202, 2014. https://eprint.iacr.org/2014/202.

[6] J. Lai, R. Deng, H. Pang, and J. Weng. Verifiable computation on outsourced encrypted data. In *Computer Security - ESORICS 2014*. 2014. (I can provide the PDF if necessary).

[7] B. Parno, C. Gentry, J. Howell, and M. Raykova. Pinocchio: Nearly practical verifiable computation. Cryptology ePrint Archive, Report 2013/279, 2013. https://eprint.iacr.org/2013/279.

[8] B. Parno, M. Raykova, and V. Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. Cryptology ePrint Archive, Report 2011/597, 2011. https://eprint.iacr.org/2011/597.

[9] P. Rauzy and S. Guilley. A Formal Proof of Countermeasures Against Fault Injection Attacks on CRT-RSA. *Journal of Cryptographic Engineering*, 2014. https://eprint.iacr.org/2013/506.

[10] P. Rauzy and S. Guilley. Countermeasures Against High-Order Fault-Injection Attacks on CRT-RSA. In *IACR Workshop on Fault Diagnosis and Tolerance in Cryptography*, 2014. https://eprint.iacr.org/2014/559.

[11] A. Shamir. Method and apparatus for protecting public key schemes from timing and fault attacks, 1999. US Patent Number 5,991,415 (https://www.google.com/patents/US5991415).