Iron: 基于Intel SGX 的函数加密系统

Ben A Fisch, Dhinakaran Vinayagamurthy, Dan Boneh, Sergey Gorbunov

斯坦福大学，滑铁卢大学
benafisch@gmail.com, dvinayag@uwaterloo.ca, dabo@cs.stanford.edu, sgorbunov@uwaterloo.ca

摘　要：函数加密(FE)是一种非常强大的加密机制，可以让授权的实体对加密数据进行计算，并且清楚地了解结果。然而，当前所有的常规FE的加密实例可行性都比较差。我们使用Intel最近的软件保护扩展(SGX)构建了一个实际可用的FE系统–Iron。我们证明，Iron可以应用于复杂的功能，甚至是实现简单功能时，表现也优于最知名的的加密方案。我们通过在硬件元件的环境中对FE建模来论证其安全性，并证明Iron满足安全模型。

1　背景介绍

Functional Encryption (FE) is a powerful cryptographic tool that facilitates non-interactive fine-grained access control to encrypted data [BSW12]. A trusted authority holding a master secret key msk can generate special functional secret keys, where each functional key sk$_f$ is associated with a function f (or program) on plaintext data. When the key sk$_f$ is used to decrypt a ciphertext ct, which is the encryption of some message m, the result is the quantity f(m). Nothing else about m is revealed. Multi-Input Functional Encryption (MIFE) [GGG+14] is an extension of FE, where the functional secret key sk$_g$ is associated with a function g that takes l⩾1 plaintext inputs. When invoking the decryption algorithm D on inputs D(sk$_g$, c$_1$, . . . , c$_l$), where ciphertext number i is an encryption of message m$_i$, the algorithm outputs g(m$_1$, . . . , m$_l$). Again, nothing else is revealed about the plaintext data m$_1$, . . . , m$_l$. Functions can be deterministic or randomized with respect to the input in both single and multi-input settings [GJKS15,GGG+14].

函数加密(FE)是一种强大的加密工具，它可以为加密的数据提供非交互式的细粒度访问控制[BSW12]。一个拥有主密钥msk的可信权限可以生成多个特殊的函数密钥，其中每个函数密钥sk$_f$与一个在纯文本数据上的函数f(或程序)相关联。当密钥sk$_f$用于解密消息m的加密密文ct时，其结果是f(m)。没有泄露关于m的其他信息。多输入函数加密(MIFE)[GGG + 14] 是FE的一个扩展，其函数秘钥sk$_g$与一个以l¿1个明文为输入的函数g相关联。在输入D(sk$_g$, c$_1$, . . . , c$_l$)上调用解密算法D，此时密文序号i是消息m$_i$的加密，算法输出是g(m$_1$, . . . , m$_l$)。同样，没有泄露关于明文m$_1$, . . . , m$_l$的其他信息。在单个和多输入设置中，函数可以根据输入选择确定性的或随机性的[GJKS15,GGG + 14]。

If FE and MIFE could be made practical, they would have numerous real-world applications. For example, consider a genetics researcher who collects public-key encrypted genomes from individuals. The researcher could then apply to an authority, such as the National Institutes of Health (NIH), and request to run a particular analysis on these genomes. If approved, the researcher is given a functional key sk$_f$, where the function f implements the desired analysis algorithm. Using sk_f the researcher can then run the analysis on the encrypted genomes, and learn the results in the clear, but without learning anything else about the underlying data.

如果FE和MIFE可实现，它们将有许多实际应用。例如，考虑一个从个体收集用公钥加密的基因组的遗传学研究人员。研究人员可以向一个授权机构提出申请，例如国家卫生研究院(NIH)，并要求对这些基因组进行特定的分析。如果获得批准，研究人员将得到一个函数密钥sk_f 。函数f可以实现所需的分析算法，使用sk_f ，研究人员可以对加密的基因组进行分析，并清楚地得到结果，但得不到其他有关基础数据的信息。

Similarly, a cloud storing encrypted sensitive data can be given a functional key sk_f, where the output of the function f is the result of a data-mining algorithm applied to the data. Using sk_f the cloud can run the algorithm on the encrypted data, to learn the results in the clear, but without learning anything else.The data owner holds the master key, and decides what functional keys to give to the cloud.

类似地，一个存储重要加密数据的云可以被赋予一个函数密钥sk_f，其中函数f的输出是对数据应用数据挖掘算法的结果。云可以使用sk_f，对加密的数据运行算法，清楚地得到结果，但得不到其他信息。数据的主人持有主密钥，并决定向云提供哪些函数密钥。

Banks could also use FE/MIFE to improve privacy and security for their clients by allowing client transactions to be end-to-end encrypted, and running all transaction auditing via functional decryption. The bank would only receive the keys for the necessary audits.

银行也可以使用FE/MIFE来提高客户的隐私和安全性，客户交易采用端到端加密，并通过函数解密运行所有事务审计。银行只会收到必要审计所需的密钥。

The problem is that current FE constructions for complex functionalities cannot be used in practice [GGH+13]. They rely on program obfuscation which currently cannot be implemented.

问题是当前为复杂功能设计的FE结构可行性差[GGH + 13]。它们基于目前还无法实现的程序混淆。

Our contribution. We propose a practical implementation of FE/MIFE using Intel’s Software Guard Extensions (SGX). Intel SGX provides hardware support for isolated program execution environments called enclaves. Enclaves can also attest to a remote party that it is running a particular program in an isolated environment, and even include in its remote attestation the inputs and outputs of computations performed by that program. Our SGX-assisted FE/MIFE system, called Iron, can run functionalities on encrypted data at full processor speeds. The security of Iron relies on trust in Intel’s manufacturing process and the robustness of the SGX system. Additionally, a major achievement of this work was not only to propose a construction of FE/MIFE using SGX, but also to formalize our trust assumptions and supply rigorous proofs of security inside this formal model.

我们的贡献：我们提出使用英特尔的软件保护扩展(SGX)实现FE/MIFE的一个可行方案。Intel SGX为孤立程序执行环境提供的硬件支持称为飞地。飞地也可以认证一个远程的组织，它在一个孤立的环境中运行一个特定的程序，甚至将由该程序执行的计算的输入和输出包含在它的远程认证书中。我们设计的由SGX作为辅助的FE/MIFE系统称为“Iron”，它可以在处理器最快运行速度下运行加密数据的函数。Iron的安全性依赖于

对英特尔制造过程的信任和SGX系统的健壮性。此外，这项工作的主要成果不仅是提出用SGX构建FE/MIFE，还可以使我们的信任假设形式化，并在此正式模型中提供严密的安全证明。

The design of Iron is described in detail in Section 3. At a high level, the system uses a Key Manager Enclave (KME) that plays the role of the trusted authority who holds the master key. This authority sets up a standard public key encryption system and signature scheme. Anyone can encrypt data using the KME' s published public key. When a client (e.g., researcher) wishes to run a particular function f on the data, he requests authorization from the KME. If approved, the KME releases a functional secret key sk_f that takes the form of an ECDSA signature on the code of f. Then, to perform the decryption, the client runs a Decryption Enclave (DE) running on an Intel SGX platform. Leveraging remote attestation, the DE can obtain over a secure channel the secret decryption key from the KME to decrypt ciphertexts. The researcher then loads sk_f into the DE, as well as the ciphertext to be operated on. The DE, upon receiving sk_f and a ciphertext, checks the signature on f, decrypts the given ciphertext, and outputs the function f applied to the plaintext. The enclave then erases all of its state from memory.

第3节详细描述了Iron的设计。在高层次上，系统使用一个密钥管理器飞地(KME)，它受信任且持有主密钥。它建立了一个标准的公钥加密系统和签名方案。任何人都可以使用KME发布的公钥加密数据。当一个客户(例如，研究人员)希望用数据运行一个特定的函数f时，他请求KME的授权。如果获得批准，KME将发布一个功能密钥sk_f，它在f的代码上表现为ECDSA签名的形式。然后，为了解密，客户在Intel SGX平台上运行一个解密飞地(DE)。利用远程认证，DE可以通过一个安全通道从KME处获得解密密钥去解密密文。然后，研究人员将sk_f以及被操作的密文加载到DE。DE在接收sk_f和密文时，检查f上的签名，解密给定的密文，并将函数f应用于明文的结果作为输出。然后这块飞地将它的所有记忆清空。

Several subtleties arise when implementing this approach. First, the specifics of SGX make it difficult to build the system as described above with a single DE that can interpret any function f. We overcome this complication by involving a third enclave as explained in Section 3. Second, we need a mechanism for the KME to ensure that the DE has the correct signature verification key before sending it the secret decryption key. Enclaves cannot simply access public information because all I/O channels are controlled by a potentially untrusted host. The simplest idea, it seems, is to have the KME send the verification key along with the secret decryption key in its secure message to the DE. However, it turns out that in order to formally prove security the message from KME to the DE also needs to be signed and verified with this verification key. We could statically code the verification key generated by the KME into the DE, but this would complicate the KME' s verification of the DE' s remote attestation. Hardcoding a fixed certified public key belonging to the KME/authority into the DE may appear to be a simple solution, but it requires an auxiliary Public Key Infrastructure (PKI) and key management mechanism, which obviates much of the KME' s role. The best option is to define the DE such that the verification key is locally loaded and incorporated into the remote attestation as a program input. Finally, there are several known side-channel attacks on SGX, and we discuss how Iron can defend against them.

在实现此方法时，会出现一些微妙之处。首先，SGX的具体细节使得我们很难像上面描述的那样用一个可以解释任何函数f的DE来构建系统。我们在第3节通过引入第三个飞地来解决这个问题。其次，KME需要机制来确保DE拥有正确的签名认证密钥，然后才发送解密密钥。飞地不能简单地访问公共信息，因为所有I/O通道都由一个可能不受信任的主机控制。最简单的想法似乎是令KME把认证密钥和自己安全信息中的解密密钥发送给DE。然而事实证明,为了正式地证明信息从KME传递到DE的安全性，也需要用认证密钥去签名和认证。我们可以将KME生成的认证密钥静态编码到DE中，但这将使KME对DE的远程认证的验证变得更加复杂。将一种固定认证的、属于KME（授权代理）的公共密钥硬件编码到DE上似乎是一个简单的解决方案，但它需要一个辅助的公钥基础设施(PKI)和密钥管理机制，这就排除了KME的大部分作用。最好的选择是定义这样的DE：认证密钥在本地加载，并将其合并到远程认证中作为程序的输入。最后，对于几种已知的针对SGX的侧向攻击，我们讨论了Iron如何防御它们。

We implemented Iron and report on its performance for a number of functionalities. For complex functionalities, this implementation is far superior to any cryptographic implementation of FE (which does not rely on hardware assumptions). We show in Section 5 that even for simple functionalities, such as comparison and small logical circuits, our implementation outperforms the best cryptographic schemes by over a 10,000 fold improvement.

我们实现了Iron，并总结了它在许多功能方面的表现。对于复杂的功能，这一实现方案远远优于任何对FE(不依赖硬件假设)的加密实现。我们将在第5 节中证明，即使是简单的功能，例如比较和小型逻辑电路，我们的实现方案比最好的加密方案要高出1万倍以上。

In Section 7 we argue security of this construction. To do so we give a detailed model of FE/MIFE security in the context of an SGX-like system, and prove that Iron satisfies the model.

2 系统设计

2.1 概述

Platforms The Iron system consists of a single trusted authority (Authority) platform and arbitrarily many decryption node platforms, which may be added dynamically. Both the trusted authority and decryption node platforms are Intel SGX enabled. Just as in a standard FE system, the trusted authority has the role of setting up public parameters as well as distributing functional secret keys, or the credentials required to decrypt functions of ciphertexts. A client application, which does not need to run on an Intel SGX enabled platform, will interact once with the trusted authority in order to obtain authorization for a function and will then interact with a decryption node in order to perform functional decryptions of ciphertexts.

平台 Iron系统由一个可信的授权平台和任意多的可以动态添加的解密节点平台组成。可信授权平台和解密节点平台都由Intel SGX 启用。就像在标准的FE系统中一样，

这个受信任的授权平台有设置公共参数，分发函数密钥和分发解密密文函数所需的认证的作用。一个不需要在Intel SGX启用平台上运行的客户端应用程序将与授权平台进行交互，以获得对一个函数的授权，然后与解密节点平台进行交互，以执行对密文的功能解密。

Protocol flow The public parameters that the Authority platform generates will consists of a public encryption key for a public key cryptosystem and a public verification key for a cryptographic signature scheme. Ciphertexts are encrypted using the public encryption key. The functional secret keys that the Authority platform issues to client applications are signatures on function descriptions. Leveraging remote attestation, the Authority platform provisions the secret decryption key to a special enclave on the decryption node. When a client application sends a ciphertext, function description, and valid signature to the decryption node, an enclave with access to the secret key will check the signature, decrypt the ciphertext, run the function on the plaintext, and output the result. The enclave will abort on invalid signatures.

协议流 授权平台生成的公共参数包括，一个公钥加密系统的加密公钥，以及用于加密签名方案的认证公钥。密文使用加密公钥加密。授权平台交给客户端应用程序的函数密钥是在函数描述上的签名。利用远程认证，授权平台将解密密钥交给在解密节点上的一个特殊的飞地。当客户端应用程序向解密节点发送密文、函数描述和有效签名时，带有认证密钥的飞地将检查签名，解密密文，在明文上运行函数，并输出结果。飞地将中止无效的签名。