Thesis Proposal

Feng-Hao Liu

September 5, 2012

Abstract

My research interests center around the area of **error-tolerant cryptography.** In cryptography, our goal is to design protocols that withstand malicious behavior of an adversary. Traditionally, the focus was on a setting where honest users followed their protocol exactly, without fault. But what if an adversary can induce faults, for example a physical attack that changes the state of a user's computation, forcing a user to accept when he should be rejecting; or tries to use a modified secret key? Can any security guarantees still be given when such errors occur? My PhD work studies the implications of various types of errors and develops techniques that protect against them.

I have delved into the following topics for different scenarios of errors: (1) cryptography with imperfect hardware, where the adversary can cause the cryptographic device to leak some secret information and tamper with the device's memory; (2) secure delegation protocols, where a user can delegate some computation to an untrusted server that causes errors.

To highlight some of my results:

- I gave a generic construction to secure *any* cryptographic functionality against continual memory tampering and leakage errors in the *split-state model* [LL12]. My main tool is to construct a non-malleable code that is also leakage resilient in this model, which resolves one central open problem in the previous work of Dziembowski et el. [DPW10].
- I developed new delegation protocols that allow a user, who only stores a short certificate of his data (potentially very large), to delegate the computation on the data to the cloud, and then verify the outcome in time *sub-linear* in the data size [CKLR11].

I elaborate on my work exploring different types of errors in cryptography in the following sections.

1 Tampering and Leakage Resilient Cryptography

Imperfect implementations of a cryptographic functionality can cause serious errors that may render the system totally insecure. Recently, the cryptographic community has been extensively studying various flavors of the following general problem. Suppose that we have a device that implements some cryptographic functionality (for example, a signature scheme or a cryptosystem). Further, suppose that an adversary can, in addition to input/output access to the device, get some side-channel information about its secret state, potentially on a continual basis; for example, an adversary can measure the power consumption of the device, timing of operations, or even read part of the secret directly [Koc96, HSH+08]. Additionally, suppose that the adversary can, also possibly on a continual basis, somehow alter the secret state of the device through an additional physical attack such as microwaving the device or exposing to heat or EM radiation [BS97, AARR02]. What can be done about protecting the security of the functionality on the device?

Unfortunately, strong negative results exist even for highly restricted versions of this general problem. For example, if the device does not have access to randomness, but is subject to arbitrary continual leakage, and so, in each round i, can leak to the adversary just one bit $b_i(s_i)$ for a predicate b_i of the adversary's choice, eventually it will leak its entire secret state. Moreover, jointly with Anna Lysyanskaya [LL10], I showed that even in a very restricted leakage model where the adversary can continually learn a physical bit of the secret state s_i, if the adversary is also allowed to tamper with the device and the device does not have access to randomness, the adversary will eventually learn the entire secret state. Further, even with tampering alone, Gennaro et al. [GLM+04] show that security from arbitrary tampering cannot be achieved unless the device can overwrite its memory; further, they show that security can only be achieved in the common reference string model.

Thus, positive results are only possible for restricted versions of this problem. If we only allow leakage, but not tampering, and access to a source of randomness that the device can use to update itself, devices for signatures and decryption can be secured in this model under appropriate assumptions [BKKV10, DHLAW10, LRW11, LLW11]. Devices that don't have access to randomness after initialization can still be secure in the more restricted bounded-leakage model, introduced by Akavia, Goldwasser, and Vaikuntanathan [AGV09], where the attacker can learn arbitrary information about the secret, as long as the total amount is bounded by some prior

parameter [AGV09, NS09, ADW09, KV09].

If only tampering is allowed, Gennaro et al. [GLM+04] gave a construction that secures a device in the model where the manufacturer has a public key and signs the secret key of the device. Dziembowski et al. [DPW10] generalized their solution to the case where the contents of the device is encoded with a non-malleable code; they consider the case where the class of tampering functions is restricted, and construct codes that are non-malleable with respect to these restricted tampering functions. Specifically, they have non-constructive results on existence of non-malleable codes for broad classes of tampering functions; they construct, in the plain model, a non-malleable code with respect to functions that tamper with individual physical bits; in the random-oracle model, they give a construction for the so-called *split-state* tampering functions, which we will discuss in detail below. improved the construction (in the plain model) of non-malleable codes that can withstand block-by-block tampering functions for blocks of small (logarithmic) sizes.

Finally, there are positive results for signature and encryption devices when both continual tampering and leakage are possible, and the device has access to a protected source of true randomness [KKS11]. One may be tempted to infer from this positive result that it can be "derandomized" by replacing true randomness with the continuous output of a pseudorandom generator, but this approach is ruled out by myself and Lysyanskaya [LL10]. Yet, how does a device, while under a physical attack, access true randomness? True randomness is a scarce resource even when a device is not under attack; for example, the GPG implementations of public-key cryptography ask the user to supply random keystrokes whenever true randomness is needed, which leads to non-random bits should a device fall into the adversary's hands.

In a subsequent work with Anna Lysyanskaya [LL12], I investigate general techniques for protecting cryptographic devices from continual leakage and tampering attacks without requiring access to true randomness after initialization. Since, as we explained above, this is impossible for general classes of leakage and tampering functions, we can only solve this problem for restricted classes of leakage and tampering functions. Which restrictions are reasonable? Suppose that a device is designed such that its memory M is split into two compartments, M_1 and M_2, that are physically separated. For example, a laptop may have more than one hard drive. Then it is reasonable to imagine that the adversary's side channel that leaks information about M_1 does not have access to M_2, and vice versa. Similarly, the adversary's tampering function tampers with M_1 without access to M_2, and vice versa.

This is known as the split-state model, and it has been considered before in the context of leakage-only [DP08, DLWW11] and tampering-only [DPW10] attacks.

Results. Let $G(\cdot,\cdot)$ be any deterministic cryptographic functionality that, on input some secret state s and user-provided input x, outputs to the user the value y, and possibly updates its secret state to a new value s'; formally, $(y, s') = G(s, x)$. For example, G can be a stateful pseudorandom generator that, on input an integer m and a seed s, generates $m + |s|$ pseudorandom bits, and lets y be the first m of these bits, and updates its state to be the next $|s|$ bits. A signature scheme and a decryption functionality can also be modeled this way. A participant in an interactive protocol, such as a zero-knowledge proof, or an MPC protocol, can also be modeled as a stateful cryptographic functionality; the initial state s would represent its input and random tape; while the supplied input x would represent a message received by this participant. A construction that secures such a general stateful functionality G against tampering and leakage is therefore the most general possible result. This is what we achieve: our construction works for any efficient deterministic cryptographic functionality G and secures it against tampering and leakage attacks in the split-state model, without access to any randomness after initialization. Any randomized functionality G can be securely derandomized using a pseudorandom generator whose seed is chosen in the initialization phase; our construction also applies to such a derandomized version of G. Quantitatively, our construction tolerates continual leakage of as many as $(1 - o(1))n$ bits of the secret memory, where n is the size of the secret memory.

Our construction works in the common reference string (CRS) model (depending on the complexity assumptions, this can be weakened to the common random string model); we assume that the adversary cannot alter the CRS. Trusted access to a CRS is not a strong additional assumption. A manufacturer of the device is already trusted to produce a correct device; it is therefore reasonable to also trust the manufacturer to hard-wire a CRS into the device. The CRS itself can potentially be generated in collaboration with other manufacturers, using a secure multi-party protocol.

As an important additional result, I constructed non-malleable codes for the split-state tampering functions in the CRS model. This improves previous results of Dziembowski et al. [DPW10], which only give a random-oracle-based construction of non-malleable codes for the split-state tampering functions; a central open problem from that paper was to construct

these codes without relying on the random oracle. Then I use this result as a building block for the main result; but it is of independent interest.

2 Secure Delegation Schemes

Delegation of computing becomes a fast growing scenario in the emergence of cloud computing – organizations or individuals (referred to as the delegator), instead of maintaining their own system, may outsource their computational tasks to specialized providers (e.g., Amazon) or anonymous users on the Internet (e.g., SETI@Home) (referred to as the cloud). In order to ensure the cloud perform the computation correctly, we like the cloud to provide a proof to the delegator, which allows the verification time significantly less than the computation time. Recently, there have been several results to this question, considering delegating general or specific functionalities, and verifying the correctness in time almost linear in the data size.

Jointly with Kai-Min Chung, Yael Kalai, and Ran Raz [CKLR11], I further consider a scenario where the data are so huge that the delegator wants to delegate them to the cloud as well. Once the data are delegated, the delegator only needs to keep a short certificate, and then can verify the computation in time sub-linear in the data size. We call this task *memory delegation*. A natural example is our email system: users store their mails in the mail server (Gmail), and they can request some computations on the mails, such as search, delete, label, etc. Another natural scenario is *streaming delegation* where a stream of huge data comes by, and the delegator, who cannot store them all, delegates the task to the cloud. Later on, the delegator may ask for some computation on the data, and a proof of correctness from the cloud. I elaborate these new two models in the following.

Memory delegation. Suppose that Alice would like to store all her memory in the cloud. The size of her memory may be huge (for example, may include all the emails she ever received). Moreover, suppose she doesn't trust the cloud. Then, every time she asks the cloud to carry out some computation (for example, compute how many emails she has received from Bob during the last year), she would like the answer to be accompanied by a proof that indeed the computation was done correctly. Note that the input to these delegated functions may be her entire memory, which can be huge. Therefore, it is highly undesirable that Alice runs in time that is proportional to this input size. More importantly, Alice doesn't even hold on to this memory anymore, since she delegated it to the cloud.

Thus, in a memory delegation scheme, a delegator delegates her entire memory to the cloud, and then may ask the could to compute functions of this memory, and expects the answers to be accompanied by a proof. Note that in order to verify the correctness of these proofs, the delegator must save some short certificate of her memory, say a certificate of size $\text{polylog}(n)$, where n is the memory size. The proofs should be verifiable very efficiently; say, in time $\text{polylog}(n, T)$, where T is the time it takes to compute the function. Moreover, Alice should be able to update her memory efficiently. We refer the reader to my paper [CKLR11] for details, and for the formal definition of a memory delegation scheme.

Streaming delegation. Suppose that there is some large amount of data that is streaming by, and suppose that a user, Alice, wishes to save this data, so that later on she will be able to compute statistics on this data. However, Alice's memory is bounded and she cannot store this data. Instead, she wishes to delegate this to the cloud. Namely, she asks the cloud to store this streaming data for her, and then she asks the cloud to perform computation on this data. As in the case of memory delegation, in order to later verify the correctness of these computations, Alice must save some short certificate of this streaming data. As opposed to the setting of memory delegation, here the certificate should be computed (and updated) in a streaming manner.

The settings of memory delegation and streaming delegation are quite similar. In both settings Alice asks the cloud to store a huge object (either her memory or the streaming data). There are two main differences between the two: (1) In the setting of streaming delegation, the certificates and updates must be computed in a streaming manner. Thus, in this sense, constructing streaming delegation schemes may be harder than constructing memory delegation schemes. Indeed, our streaming delegation scheme is more complicated than our memory delegation scheme, and proving soundness in the streaming setting is significantly harder than proving soundness in the memory setting. (2) In the setting of streaming delegation, the memory is updated by simply adding elements to it. This is in contrast to the setting of memory delegation, where the memory can be updated in arbitrary ways, depending on the user's needs. However, in the memory setting, we allow the delegator to use the help of the worker when updating her certificate (or secret state), whereas in the streaming setting we require that the delegator updates her certificate on her own. The reason for this dis-

crepancy, is that in the memory setting the delegator may not be able to update her certificate on her own, since she may want to update her memory in involved ways (such as, erase all emails from Bob). On the other hand, in the streaming setting, it seems essential that the delegator updates her certificate on her own, since in this setting the data may be streaming by very quickly, and there may not be enough time for the delegator and worker to interact during each update.

Results. I construct both memory delegation and streaming delegation schemes. The memory delegation scheme consists of an offline phase, where the delegator D delegates her memory $x \in \{0,1\}^n$ to a worker W. This phase is non-interactive, where the delegator sends a single message, which includes her memory content x to the worker W. The runtime of both the delegator and the worker in the offline phase is $\text{poly}(n)$, where n is the memory size. At the end of this phase, the delegator saves a short certificate σ of her memory, which she will later use when verifying delegation proofs.

The streaming delegation scheme, on the other hand, doesn't have such an offline phase. In the streaming setting, we consider the scenario where at each time unit t a bit x_t is being streamed. The delegator starts with some secret state (or certificate) σ_0, and at time unit $t+1$ she uses her secret state σ_t and the current bit x_{t+1} being streamed, to efficiently update her secret state from σ_t to σ_{t+1}.

In both settings, each time the delegator D wants the worker W to compute a function $f(x)$, they run a delegation protocol, which we denote by $\mathsf{Compute}(f)$. The memory delegation scheme also has an Update protocol, where the delegator D asks the worker W to update her memory and to help her update her secret state σ. The latter can be thought of as a delegation request, and the guarantees (in term of runtime and communication complexity) are similar to the guarantees of the $\mathsf{Compute}$ protocol.

In the streaming setting, the delegator updates her secret state on her own in time $\text{polylog}(N)$, where N is an upper bound on the length of the stream. Namely, the update function, that takes as input a certificate σ_t and a bit x_{t+1}, and outputs a new certificate σ_{t+1}, can be computed in time $\text{polylog}(N)$.

Our delegation protocol $\mathsf{Compute}(f)$ is non-interactive (i.e, consists of two messages, the first sent by the delegator and the second sent by the worker). It is based on the non-interactive version of the delegation protocol of Goldwasser *et. al.* [GKR08, KR09], denoted by GKR (though is significantly more complicated than merely running GKR). In what follows

I state the theorems formally, and refer the readers to my paper [CKLR11] for the formal definitions and models.

Theorem 1 (Memory Delegation) *Assume the existence of a poly-log PIR scheme and assume the existence of a collision resistant hash family. Then there exists a non-interactive (2-message) memory delegation scheme* mDel*, for delegating any function computable by an* $\mathcal{L}$*-uniform poly-size circuit. The delegation scheme,* mDel *has the following properties, for security parameter* k*.*

- *The scheme has perfect completeness and negligible (reusable) soundness error.*
- *The delegator and worker are efficient in the offline stage; i.e., both the delegator and the worker run in time* $\mathrm{poly}(k, n)$*.*
- *The worker is efficient in the online phase. More specifically, it runs in time* $\mathrm{poly}(k, S)$ *during each* $\mathsf{Compute}(f)$ *and* $\mathsf{Update}(f)$ *operation, where* S *is the size of the* $\mathcal{L}$*-uniform circuit computing* f*. The delegator runs in time* $\mathrm{poly}(k, d)$ *during each* $\mathsf{Compute}(f)$ *and* $\mathsf{Update}(f)$ *operation, where* d *is the depth of the* $\mathcal{L}$*-uniform circuit computing* f*.*[1]

In particular, assuming the existence of a poly-logarithmic PIR scheme, and assuming the existence of a collision resistent hash family, we obtain a memory delegation scheme for $\mathcal{L}$-uniform **NC** computations, where the delegator D runs in time *poly-logarithmic* in the length the size of the memory.

Theorem 2 (Streaming Delegation) *Let* k *be a security parameter, and let* N *be a parameter (an upper bound on the length of the stream). Let* $\mathcal{F}$ *be the class of all* $\mathcal{L}$*-uniform poly-size boolean circuits. Assume the existence of a fully-homomorphic encryption scheme secure against* $\mathrm{poly}(N)$*-size adversaries. Then there exists a streaming delegation scheme* $\mathsf{sDel}_{\mathcal{F}}$ *for* $\mathcal{F}$ *with the following properties.*

- $\mathsf{sDel}_{\mathcal{F}}$ *has perfect completeness and negligible reusable soundness error.*
- D *updates her secret state in time* $\mathrm{polylog}(N)$*, per data item.*
- *In the delegation protocol, when delegating a function* $f \in \mathcal{F}$ *computable by an* $\mathcal{L}$*-uniform circuit of size* S *and depth* d*, the delegator* D *runs in time* $\mathrm{poly}(k, d, \log N)$*, and the worker* W *runs in time* $\mathrm{poly}(k, \log N, S)$*.*

[1] Thus, for every constant $c \in \mathbb{N}$, if we restrict the depth of f to be at most k^c, then the delegator is considered efficient.

In particular, assuming the existence of a fully-homomorphic encryption scheme secure against adversaries of size poly(N), we obtain a streaming delegation scheme for $\mathcal{L}$-uniform **NC** computations, where the delegator D runs in time *poly-logarithmic* in the length of data stream.

My Publications

Papers related to this proposal:

- [LL12] with Anna Lysyanskaya. "Tamper and Leakage Resilience in the Split-State Model." In Crypto 2012. (Sec 1.)
- [LL10] with Anna Lysyanskaya. "Algorithmic tamper-proof security under probing attacks." In SCN 2010. (Sec 1.)
- [CKLR11] with Kai-Min Chung, Yael Kalai, and Ran Raz. "Memory Delegation" In Crypto 2011. (Sec 2.)

Papers in other directions:

- [HLY12] with Yun-Ju Huang Bo-Yin Yang. "Public-key Encryption Schemes from Multivariate Quadratic Assumptions" In PKC 2012.
- [YCCL11] with Ching-Hua Yu, Sherman Chow, and Kai-Min Chung. "Efficient Secure Two-Party Exponentiation." In RSA 2011.
- [CLLY10] with Kai-Min Chung, Chi-Jen Lu, and Bo-Yin Yang. "Efficient string-commitment from weak bit-commitment." In Asiacrypt 2010.
- [CL10] with Kai-Ming Chung. "Tight parallel repetition theorems for public-coin arguments." In TCC, 2010. **Recipient of Best Student Paper Award.**
- [LLY08] with Chi-Jen Lu and Bo-Yin Yang. "Secure PRNGs from Specialized Polynomial Maps over Any $\mathbb{F}_q$." In PQcrypto 2009.

References

[AARR02] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The em side-channel(s). In *CHES*, volume 2523, pages 29–45. LNCS, 2002.

[ADW09] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In 5677, editor, *CRYPTO*, pages 36–54. LNCS, 2009.

[AGV09] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC*, volume 5444, pages 474–495. LNCS, 2009.

[BKKV10] Zvika Brakerski, Yael Tauman Kalai, Jonathan Katz, and Vinod Vaikuntanathan. Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. In *FOCS*, pages 501–510. IEEE, 2010.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *CRYPTO*, volume 1294, pages 513–525. LNCS, 1997.

[CKLR11] Kai-Min Chung, Yael Tauman Kalai, Feng-Hao Liu, and Ran Raz. Memory delegation. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 151–168. Springer, 2011.

[CL10] Kai-Min Chung and Feng-Hao Liu. Parallel repetition theorems for interactive arguments. In Daniele Micciancio, editor, *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 19–36. Springer, 2010.

[CLLY10] Kai-Min Chung, Feng-Hao Liu, Chi-Jen Lu, and Bo-Yin Yang. Efficient string-commitment from weak bit-commitment. In Masayuki Abe, editor, *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 268–282. Springer, 2010.

[DHLAW10] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Cryptography against continuous memory attacks. In *FOCS*, pages 511–520, 2010.

[DLWW11] Yevgeniy Dodis, Allison B. Lewko, Brent Waters, and Daniel Wichs. Storing secrets on continually leaky devices. In *FOCS*, pages 688–697, 2011.

[DP08] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302. IEEE, 2008.

[DPW10] Stefan Dziembowski, Krzysztof Pietrzak, and Daniel Wichs. Non-malleable codes. In *ICS*, pages 434–452, 2010.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC*, pages 113–122, 2008.

[GLM+04] Rosario Gennaro, Anna Lysyanskaya, Tal Malkin, Silvio Micali, and Tal Rabin. Algorithmic tamper-proof (atp) security: Theoretical foundations for security against hardware tampering. In *TCC*, volume 2951, pages 258–277. LNCS, 2004.

[HLY12] Yun-Ju Huang, Feng-Hao Liu, and Bo-Yin Yang. Public-key cryptography from new multivariate quadratic assumptions. In *Public Key Cryptography*, pages 190–205, 2012.

[HSH+08] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys. In *USENIX Security Symposium*, pages 45–60, 2008.

[KKS11] Yael Tauman Kalai, Bhavana Kanukurthi, and Amit Sahai. Cryptography with tamperable and leaky memory. In *CRYPTO*, volume 6841, pages 373–390. LNCS, 2011.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, volume 1109, pages 104–113. LNCS, 1996.

[KR09] Yael Tauman Kalai and Ran Raz. Probabilistically checkable arguments. In *CRYPTO*, 2009.

[KV09] Jonathan Katz and Vinod Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, volume 5912, pages 703–720. LNCS, 2009.

[LL10] Feng-Hao Liu and Anna Lysyanskaya. Algorithmic tamper-proof security under probing attacks. In *SCN*, volume 6280, pages 106–120. LNCS, 2010.

[LL12] Feng-Hao Liu and Anna Lysyanskaya. Tamper and leakage resilience in the split-state model, 2012.

[LLW11] Allison B. Lewko, Mark Lewko, and Brent Waters. How to leak on key updates. In *STOC*, pages 725–734. ACM, 2011.

[LLY08] Feng-Hao Liu, Chi-Jen Lu, and Bo-Yin Yang. Secure prngs from specialized polynomial maps over any. In *PQCrypto*, pages 181–202, 2008.

[LRW11] Allison B. Lewko, Yannis Rouselakis, and Brent Waters. Achieving leakage resilience through dual system encryption. In *TCC*, volume 6597, pages 70–88. LNCS, 2011.

[NS09] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO*, volume 5677, pages 18–35. LNCS, 2009.

[YCCL11] Ching-Hua Yu, Sherman S. M. Chow, Kai-Min Chung, and Feng-Hao Liu. Efficient secure two-party exponentiation. In *CT-RSA*, pages 17–32, 2011.