tiCrypt White Paper

Framework, Security Overview, and Compliance

Thomas Samant and Alin Dobra

Tera Insights, LLC

June 19, 2017

Executive Summary

Judging by the continuous stream of data breaches, it seems that the security mechanisms widely deployed such as back-end network security, perimeter defenses, and security permission management, are not effective. What these mechanisms all fail to address is the possibility that an administrator account becomes compromised, granting access to all data on the server. This seems to be a fundamental impasse and creates crisis in the cybersecurity sector.

The primary assumption behind *ticrypt*'s design is that back-end servers and network infrastructures can be easily compromised. The only mechanism that *tiCrypt* trusts and what it builds the security of the system on is strong encryption controlled by the user. Since the decryption keys are not stored on the server, the data cannot be compromised if the server is breached. In *tiCrypt* , a compromised admin account does not translate into a data breach. For more questions or more information please email us at: info@terainsights.com.

tiCrypt **Security Principles**

- **Data Confinement through Encryption:** Rather than a direct hardening of the system through traditional means, *tiCrypt* utilizes encryption to ensure data confinement. System resources such as files, drives, and communication channels use an independent AES-256 encryption key.
- **Public-Key Cryptography:** Data confinement requires independent keys for each resource. The key management nightmare is avoided in *tiCrypt* through the use of public key cryptography. Each user, upon account creation, creates an RSA-2048 key; this key is the primary means of secure data control.
- **End-to-end encryption for security:** This is defined as encrypting a file at the earliest stage and decrypting it as late as possible. *tiCrypt* heavily relies on end-to-end encryption to guarantee that data is maximally protected. This ensures that even system administrators do not have access to user's data.

Advanced Security Features

- **Public Key Signing and Chain of Trust:** All public keys are digitally signed to ensure authenticity.
- **Crowd Security:** The client remembers fingerprints of public keys to prevent user impersonation.
- **Key Escrow:** Advanced key recovery mechanism based on multi-party authentication.
- **Secure Virtual Machines:** Secure data processing using encrypted drives, connection to user data and SSH tunnels.

Auditing and Monitoring Features

- **Advanced Audits:** *tiCrypt* provides detailed, efficient, auditing and monitoring capabilities. By using our custom developed large data engine, capable of processing hundreds of millions of entries per seconds, even large audit logs can be mined efficiently for usage patterns, compliance, and more.
- **Compliance and Regulations:** *tiCrypt* was built to meet the needs of organizations with sensitive data by complying with the major regulatory departments (HIPAA, FISMA, FIPS).

1 Introduction

Across the globe, reports of hacks and data breaches have become commonplace in media headlines. From Fortune 500 companies to small businesses, it seems that no business is immune. In the last two years alone, high profile companies, with security budgets upwards of $150 million, have seen their systems compromised and data breached. It seems that no amount of armed guards, sophisticated locks, firewalls or intrusion detection systems effectively protect the system.

Traditional security systems primarily focus on three types of mechanisms: back-end network security (VLANs, secure networks, etc.), perimeter defense (intrusion detection, firewalls, etc.), and security permission management (access control lists, business rules, etc.). The hope is that by keeping the perimeter secure, the outside attacks are prevented. By employing sophisticated permission management, the intent is to confine insiders to their data. The problem is that the perimeter defenses can be defeated or avoided altogether by an inside attack. Once an admin account is compromised, the perimeter defense and permission management offer no protection; there is little stopping the attackers from accessing *all* the information within the perimeter. This reaffirms that traditional data security models, including perimeter defense, are fundamentally flawed. Each occurring breach, compromised server, or hijacked admin account mounts evidence against the traditional model.

Unfortunately, despite evidence mounting against the traditional model, businesses are reluctant to learn from their mistakes. The go-to response after a hack is to improve and strengthen security features to prevent the occurrence. This mentality evokes the unending cycle of poor solutions. To truly correct this flaw and give companies the best chance of avoiding another incident, rethinking the fundamental approach to security is essential.

The one thing the hackers (and arguably even governments) are not capable of breaching is strong encryption. If the data on the server is encrypted at all times with keys controlled by individual users, a data breach would reveal very little to an attacker. For this reason, the primary defense mechanism in *tiCrypt*,is the reliance on user controlled end-to-end encryption, not server security. When coupled with traditional defense mechanisms, *tiCrypt* offers a much more secure solution to data storage, sharing and processing. In effect, an attacker has to defeat both the encryption-based and the perimeter- based defenses to successfully steal data – a much harder task.

Encryption on modern systems: While encryption has been long recognized as a good solution for data security, encryption is believed to be too inefficient for most situations, particularly on the client. With the implementation in hardware of AES-256 algorithm on both Intel and AMD processors, encryption and description are greatly accelerated. Speeds of 200-300 MB/second are possible on laptops and 1GB/second on servers. *tiCrypt* takes great advantage of these advances and incurs no significant penalty for the extensive use of encryption. Since encryption is feasible on the client, the server is unburdened by the need to perform such operations across the entire usage space. Furthermore, from a security standpoint, client encryption is highly desirable since the server is the primary hacking target.

Key management using *public key cryptography*: One of the main problems when employing encryption is how to manage the encryption keys of various components. If the user is required to remember passwords for each independent entity that needs to be encrypted, key management becomes a very serious problem. Even more importantly, secure sharing requires a secure hand-off of encryption keys to another party. Sending such keys through other means (email, phone, etc.) would significantly undermine the security of the system. The elegant solution *tiCrypt* uses is based on public key cryptography: each user, at account creation time, creates a public-private key pair. Secure hand off of encryption keys to a second party is as easy as encrypting the resource key with the public key of the intended data recipient. Using public-private key pairs for all participants can solve a much larger set of problems in a system like *tiCrypt* . In fact, all security features in *tiCrypt* are rooted in the use of public key cryptography.

Secure components vs. End-to-end encryption: When it comes to system security, the traditional approach is to secure each component independently in the hope that this will translate into good overall security. Typical security terms are *network security* (securing the network such that an attacker cannot snoop on the transmitted data), *encryption-at-rest* (securing the stored data through encryption so that storage devices cannot be accessed if removed from the system), and *application security* (securing the data while used by the application so that the information cannot be accessed by an attacker watching the application behavior). Security problems arise at the junction points and, more importantly, can be compromised by the users controlling the juncture points. *tiCrypt*'s approach is to use end-to-end encryption: data is encrypted by the user as early as possible and decrypted by the intended destination. No other participant including admins and hackers, is able to decrypt the data whether it is in transit, stored, or used by the application.

Strongly encrypt all data: A classic mistake is to allow multiple levels of security for the data. In *tiCrypt*,all data has the same security level, equivalent to *top security*. While this seems restrictive, powerful sharing mechanisms in *tiCrypt* allow easy sharing with large number of other participants. Since differences in performance for performing strong encryption are small (20% speed difference between AES-128 and AES-256), there is no practical reason to downgrade security even for low security resources.

The above ideas allow *tiCrypt* to provide a very strong form of *data confinement*. The access to each file, drive or message is fully under the control of the user that created the resource and independent of all other resources in the system. Without a cryptographically secure key exchange initiated by the data owner, nobody (including the system administrators) can effectively access the data. When coupled with advanced auditing facilities in *tiCrypt*,this property allows confinement and deep knowledge of any data leak. In the rest of this document we discuss in greater detail properties of various mechanism deployed in *tiCrypt* to ensure data security and confinement.

2 Security Principles

tiCrypt is based on three security principles: data confinement, use of public-key cryptography, and end-to-end encryption. The combination of these principles is at the core of all *tiCrypt* security features.

2.1 Data Confinement Through Encryption

The key idea behind *data confinement* is to isolate resources from each other. In particular, knowledge about how to access a resource should not give any indication on how to access another. The traditional way to enforce data confinement is through the use of *access control lists* (ACL). The main drawback of ACLs is the fact that administrators have full control over them and can bypass any ACL based mechanisms. Unfortunately, hackers that impersonate admins have the same power, thus effectively breaking data confinement.

tiCrypt primarily employs cryptographic methods to enforce data confinement. Each file, upon creation, gets encrypted with a unique, independently generated, AES-256 key. These keys get encrypted with the public keys of the users (and groups) that have access to the file and are then stored on the server. Using end-to-end encryption, the encryption is performed before moving the data onto the server while the decryption only takes place at the destination. Existence of an encrypted key on the server serves now the role of the ACL but, more importantly, the file key can be recovered only if the *private key* is used.

File sharing is performed by a secure key exchange in the browser of the file owner (as opposed to a server operation). First, the encrypted key is accessed form the server. Second the encrypted key is decrypted using the owner's private key. Then the key is encrypted with the destination's public key. Last, the key encrypted with the destination's public key is deposited on the server. Notice that this mechanism always keeps the private key on the owner's side; only public keys are available to all other users.

2.2 Public Key Cryptography

The Definition of Public Key Cryptography

Public key cryptography uses two keys per user: a public key, which is known to everyone, and a private key known only to the particular user. For example, if user Tom wants to share a message with Jim, he uses Jim's public key to encrypt the message and send it securely. Decrypting the message requires knowledge of the private key, which only Jim has, making him the only user able to read the message. Until 1977, such public-private key schemes were believed to be impossible. Through pioneering work of Rivest, Shamir and Adleman such cryptographic systems were introduced. Mathematically, a public key cryptosystem needs a core problem that is easy to solve in one direction but hard in the other. In the case of RSA, multiplying primes is relatively easy but factoring a product of large primes is very hard. The public key is a fixed exponent e and the product of the primes $n = pq$. The private key is $d = \frac{(p-1)(q-1)}{e}$ and n. The public and private keys are opposites of each other: messages encrypted with one have to be decrypted with the other. Specifically, for m a message, $m^e \bmod n$ is the encrypted message; it can be decrypted by performing $d^{(m^e)} \bmod n = m \bmod n$. Conversely, $d^m \bmod n$ is a digital signature that can be verified by performing $e^{(d^m)} \bmod n = m \bmod n$.

Beyond the mathematical details, when using any public key system two rules need to be remembered. When data is first encrypted with the public key and decrypted with the private key, we call the process encryption. When data is first encrypted with the private key but decrypted with the public key, we call the process digital signature. *tiCrypt* makes use of both mechanism to achieve both secrecy and authentication.

Using Public Key Encryption

Public key encryption is at the core of all security mechanisms used within the modern day web. From banking to social media, parts or all of its principles are seen in use. Public key encryption principles allow:

- Checking the authenticity of a website through SSL certificates
- Establishing a secure communication channel with the web server
- Digitally signed and certified messages

The use of public key encryption removes the need for passwords or secrets to authenticate users, establish secure communication, or sign messages. The private key of the pair is now the means to achieve security. The public key of the pair is made widely available and stored in plain text on servers, client computers or even business cards. The magic of public key encryption is that everybody can encrypt using the public key but only the private key holder can decrypt. Conversely, documents can be digitally signed using the private key, where the authenticity of the signature can be verified with the public key.

Of course, the larger the key, the more secure the data encrypted with that key is. It is estimated that it would take half a million desktop computers running for 13 billion years to derive the private key from an RSA-2048 public key. This is the type of key each user has within *tiCrypt* and the type recommended for government use by the NIST FIPS 140.2 publication.

Keys Allow for Features

Since public key encryption is versatile, it makes sense to give a key to each individual user. This allows users to securely do the following:

- **Authenticate/Login:** Authentication occurs without a password. The server uses the public key to verify the user's identity. A random challenge number is generated, encrypted with the public key, and sent to the user. To prove possession of the private key, the challenge has to be correctly decrypted, and the response must be properly re-encrypted with that same private key.

- **Share Files:** Sharing occurs without a file password/secret. Encrypting the file's symmetric key with the other user's public key ensures that only the recipient can decrypt their key to the file.
- **Issue Certificates:** Certificates are a list of instructions digitally signed using the private key. They entitle other users to perform operations on the owner's behalf.

2.3 End-to-end Encryption

End-to-end encryption refers to the security principle that requires the data to be encrypted as soon as possible and decrypted as late as possible. Only the intended recipient of the data should have the decryption keys and should perform the decryption just before they plan to use the data. In this form, end-to-end encryption ensures maximum security of the data since it reduces to a minimum the opportunity for an attacker.

- Since *tiCrypt* encrypts each file independently with a key controlled by the user, end-to-end encryption is possible and fully under user control.
- Man-in-the-middle attacks, otherwise known as interception attacks, are prevented independently of SSL. Even if the SSL certificate is compromised, the communication is still secure.

End-to-end encryption is strictly more secure than encryption in transit because the files are not decrypted until the verified recipient views them. Encryption in transit allows admins and any hacker who has access to the proper certificates to decrypt the files. In all cases, *tiCrypt* implements end-to-end encryption, even if the file is accessed or shared multiple times. Should a file be needed multiple times, it is decrypted multiple times with no copies residing in unsecured storage, each time implementing end-to-end encryption.

3 Advanced Security Features

tiCrypt has a number of advanced security features that address sophisticated attacks and allow key recovery under strict situations.

3.1 Public Key Signing and Chain of Trust

While the use of public key cryptography such as RSA-2048 ensures significant hardening of the system, there is still a significant vulnerability an attacker can exploit: exchanging the public key of a user with a public key controlled (i.e. for which the private key is known) by the attacker. Should a file be shared with the *fake user*, the data can be decrypted by the attacker. This attack requires a compromise of the server and only applies to future file sharing (files shared with the correct key cannot be decrypted), but it is a vulnerability nonetheless. As a preventive measure, *tiCrypt* uses digital signatures in the following manner to ensure that *fake keys* cannot be substituted:

- The system uses a *site key* that is itself signed by a Tera Insights private key verifiable with a hard-wired (in both the front-end and back-end) public key. The site key is used to bootstrap the security of the entire software deployment. The public part of the site key is readily available to all parts of the system. Its integrity can be checked by checking the digital signature using the hard-wired Tera Insights public key.
- The first super-admin key is signed by the signed key upon creation. Its integrity can be checked using the site key.

- Any new account needs to be approved by an administrator. Upon approval, the public key associated with the account is signed by the individual approving (using their private key), thus cryptographically proving the account is valid. This effectively forms a *chain of trust* for each public key linking all the way to the hard-wired Tera Insights key.
- At any point in time, any user can require the digital signatures for the entire chain of trust and validate, on the client side, each key along the way. An attacker cannot inject a *fake key* unless it controls an administrator key.

3.2 Crowd Security

To further strengthen the public key integrity, *tiCrypt* employs a form of *crowd security*. In particular, the front-end (usually the browser) will store locally, out of reach of the system administrators, *fingerprints* of the public keys of users previously interacted with. When a file is shared with a user, *tiCrypt* front-end will request the public key of the user from the server and check that the fingerprint is the same. *tiCrypt* uses SHA-256 as a secure fingerprinting technique, which is a NIST FIPS 140.2 recommended secure hashing scheme.

The chain of trust combined with crowd security provide a formidable mechanism that is hard for any attacker to defeat. Any irregularity in the use of keys is likely to be discovered in a short amount of time, allowing any possibility for a leak to be removed.

3.3 Secure JavaScript Delivery

For the purpose of high-availability across platforms, the *tiCrypt* frontend is web based and employs technologies such as: HTML5, AngularJS, ASM-JS, etc. While this provides a modern, highly functional frontend, the use of web technologies opens a potential significant vulnerability: injecting malicious JavaScript code that steals private keys.

tiCrypt uses multiple mechanisms to mitigate the risk of compromised JavasScript:

- The *tiCrypt* front-end bundles up all dependencies and does not refer to outside JavaScript, CSS or HTML resources. Compromising online JavaScript libraries would have no effect on packaged *tiCrypt* frontends.
- The *tiCrypt* front-end bundle is delivered in the form of a digitally signed Zip archive. The digital signature keys are controlled by two top level Tera Insights technology executives. This ensures the integrity and security of the JavaScript code.
- A special Java program, `ti-launcher.jar` (also digitally signed), can be used to securely deliver the front-end locally. Upon loading the Zip archive to memory, `ti-launcher.jar` will check the digital signature to make sure it is not compromised. Since the ti-launcher provides extra functionality and integration with the local host, it is the preferred way to deploy *tiCrypt* frontends.

3.4 Key Escrow

Since loss of private key translates in loss of data access in *tiCrypt*,a key recovery mechanism – *key escrow* – is needed to prevent data loss. In contrast, a key recovery mechanism that is too easy is a big security risk. In *tiCrypt*,the key recovery using the escrow mechanism is designed to require the cooperation of multiple people to protect against individual key loss. Specifically, *tiCrypt* requires a minimum of three *escrow keys* to be used. These escrow keys need to be signed with the site key (not with any of admin keys) and can be used exclusively for escrow activities (i.e. not for regular user activities). A user key is placed in escrow using the following mechanism:

1. When the user key needs to be (re)escrowed, the escrow keys with their digital signatures are downloaded from the server.
2. Digital signatures of all escrow keys are verified against the site key (which is itself previously verified against the hard-wired key).
3. An AES-256 random key is generated for each escrow key and then encrypted with the corresponding escrow public key. These encrypted keys are stored on the server.
4. The combination of the AES-256 random keys, one for each escrow key, is used to encrypt user's private key and then stored on the server

In order to recover user's private key, all escrow key holders have to securely transfer their part of the key to a designated party. An important part of this scheme is the fact that the private keys cannot be recovered if any part is missing. Moreover, none of the escrow key holders are able to recover the key – only the designated party can.

3.5 Secure Virtual Machines

tiCrypt allows secure computation through the use of *secure virtual machines*. The specific mechanism employed is fundamentally different than existing cloud solutions. In particular:

- **Securing at start-up:** The VM controller will *secure* the VM during start-up sequence by changing all passwords, disabling any SSH accounts, and blocking any ports except SSH/22 (for future use).
- **Accessing VMs:** For security reasons, virtual machines in *tiCrypt* cannot be accessed using any of the traditional methods. In particular, direct connections to the VM console, SSH logins, or other remote server technology is not allowed since it can be controlled by the admins, and thus by hackers impersonating them. The only means of communication is through a secure channel that only the *owner* of the VM can connect to:
  - When a VM starts, it gets the public key of the owner. The owner is told the public key of the VM.
  - During secure channel setup, the VM and the user co-authenticate each other using the respective public and private keys
  - A secret channel key is negotiated (Diffie-Helman protocol) and used for the communication.
- **Encrypted Drives:** In order for the VM to perform efficient computation, it needs access to fast, secure storage. This is provided in the form of *encrypted drives*. Using encrypted drives, regular applications are secured since their input, output, and intermediate results are now protected by cryptography.

  The drive encryption keys are generated and manipulated in the same manner as the file keys. The user-VM encrypted channel is used to transmit such keys to the VM during the start-up sequence. This way, the user is in full control of the drive encryption; even admins are prevented from accessing the data.
- **Access to the data store:** The VMs can access the main data store through secure sub-sessions created on the client and file keys passed through the encrypted user-VM channel.
- **Secure SSH Tunnels:** The VM controller, using the user-VM secure channel, can set up SSH tunnels between the user's computer and the virtual machine. The SSH tunnel uses exclusively public-key based login and temporary private keys. All traffic between a user's computer and virtual machine is sent through this tunnel. Effectively, any communication between is now possible with no possibility of snooping by any other party (including the system administrator).

- textbfRestricted outbound Internet access: The VMs have severely restricted internet access: only the tiCrypt backend and licensing servers can be accessed directly. The main reason for this is to protect against accidental or intentional information leaks. Even if a VM is hijacked, the only way to leak information is through the main vault, which is audited.

Using secure virtual machines in *tiCrypt*,it is possible to run and securely communicate to remote versions of any Windows or Linux distributions. The virtual machines are cryptographically isolated from the outside world; only the user's machine has access.

4 Combining *tiCrypt* with traditional defense mechanisms

Since *tiCrypt* is a software-based solution, it can be easily combined with traditional defense mechanisms like multi-factor authentication and perimeter defenses in order to provide extra layers of protection. An attacker needs to defeat all these mechanisms for a data breach, a much harder task.

4.1 Multi-factor authentication

While *tiCrypt* has very strong security, leaked private keys can still result in loss of the information covered by the key. The attacker can use the stolen key to impersonate the user and access all user files. As an added layer of protection, extra authentication factors that are independent of *tiCrypt* keys and the passwords protecting them can be added to the system. Since, typically, organization-wide authentication or token-based second factor authentication is already in place, *tiCrypt* can leverage the existing mechanism as an extra authentication factor.

The main mechanism to accommodate extra authentication factors is to detect when the private key is used in an unusual way. Specifically, if a key is used from a machine that was inactive for a set period of time by the same user or was never used, the *tiCrypt* authentication protocol will require the use of the extra factors to independently prove the authenticity of the request. If a private key is stolen and used from another computer, the second factor mechanisms will catch that and still deny access.

4.2 Perimeter and server defense

First, since *tiCrypt* deployment does not require regular user accounts on the hosting server, most server-side vulnerabilities are eliminated. Only system administrators need access to the server, thus access can be highly restricted. Moreover, strong authentication based on SSH public keys can be deployed for management purposes, thus rendering most password based vulnerabilities ineffective.

Second, the server running the *tiCrypt* back-end requires only two entry points: the SSH access for management used exclusively by the administrators, and Https access for the *tiCrypt* interface. All other ports can and should be blocked to further reduce the attack surface of the server.

Third, *tiCrypt* is a software based solution with encryption controlled by the client. It does not require special security appliances or special setups on the server. All existing security mechanisms like firewalls, intrusion detection, system auditing, etc are compatible with *tiCrypt* and can be used to strengthen the server. Organizations can leverage all existing expertise in system security to add these extra layers of protection.

5 Threat Model and Solutions

With current events running rampant with news on cybersecurity attacks and hacks, we have researched cybersecurity breaches from 2007-2015 and analyzed why a specific breach was able to occur and how *tiCrypt* use would have prevented it.

Problem: Server Misconfiguration

Over a short period of time, one or multiple servers are misconfigured, not using two-factor authentication, or left unsecured. There are no formal systems to check for mis-configurations. A server rebooting improperly is often enough to cause this problem.

Solution

In *tiCrypt*,if a server is misconfigured, it will at most lock the system. Breaking into the back-end servers provides no benefit and is not useful in any way to an attacker. Even if the authentication fails completely (for example the authentication server somehow forgets to authenticate), the accessible data is useless without the private keys that are held only by the various users. In no scenario can *tiCrypt* leak present or future unencrypted data.

Problem: Insider Threats

If admins have access to all the data, any of them can steal before being noticed. Audit logs are useless since they require too many man hours to review and keep up-to-date. Unless unforgeable audits are used for files, traditional logs will not effectively serve as a deterrent. Even if everything is logged, an extremely good solution for log analysis must be in place. Otherwise, it will be months before any rogue activity is noticed. Edward Snowden managed to steal tens of thousands of documents from a system with higher security than FISMA, all because he was a systems admin.

Solution

In *tiCrypt*,it is trivial to isolate the system administrators from the normal users. We can formally check, based on the audit log, that the system administrators do not have access to any user information. Since any legitimate access happens through the *tiCrypt* system, log mining will reveal any irregularity within minutes.

Problem: Authentication attacks

The authentication server becomes another point of failure. If data in the system is valuable enough (medical data is valued at $50-500 per medical record), protecting the server becomes a serious problem. Using fake authentication certificates, anybody can be impersonated without any actual password cracking.

Solution

In *tiCrypt* all public keys are digitally signed and fingerprints of public keys are stored on the client for all users. An attacker cannot switch keys in an attempt to trick users to share information using keys controlled by the attacker. The fingerprinting is particularly hard to fool since this data is stored client-side, outside the server's reach.

Notice that a user attempting to share information with another user can *validate* the key of the destination even with a compromised server. The validity of the trust chain can be established independently of the server and cannot be compromised without breaking public key cryptography.

Problem: Zero Day Exploits

If zero-day exploits, which are undisclosed vulnerabilities that could be exploited, are found for any of the servers, data on that server can be stolen. If such exploits are found for the authentication server, data on all servers can be stolen. While audit logs help identify what was compromised, they do nothing to prevent the attack. If the security is based on *foolproof* authentication alone, this becomes very problematic. If the data on the server is valued at upwards of $10 million (as it often is when medical data is stored on the server), such high-end attacks are probable.

Solution

tiCrypt thwarts all back-end attacks using end-to-end encryption, even if attackers controlled the servers for months. This is due to the fact that the decryption keys are not available on the server; the server is never involved in either encryption or decryption of data. At worst, a breach of the server makes the system unavailable but not insecure.

6 *tiCrypt* 's Compliance and Monitoring

tiCrypt is built to be HIPAA, FISMA, and FIPS compliant. With the use of our high powered data engine, *tiCrypt*'s audit logs can be mined and reviewed for irregularities and suspicious activities. 30+ types of events are tracked at the millisecond time stamp level to ensure a complete record of all actions.

Compliance within *tiCrypt* *tiCrypt* was built to meet the needs of organizations with sensitive data. *tiCrypt* adheres to HIPAA, FISMA, NIST 800-71 and FIPS compliance requirements where appropriate. It is FIPS 140-2 certified in the Mozilla browser with FIPS mode enabled in settings. *tiCrypt* utilizes guidelines for NIST 800-71 for access control, identification and authentication, and audit and accountability. For more information on compliance please contact info@terainsights.com.

Secure Log Authentication While it is important to protect files against rogue employees and outside attacks, it is equally as important to provide an audit trail of who accessed what information. While basic audits are mandated by most regulatory agencies, *tiCrypt* goes beyond this minimum requirement and offers detailed, secure audits combined with forensic tools to analyze the logs.

Securing the Log Using secure hashing (SHA-256), log entries are chained much like Bitcoin transactions. Knowledge of the hashes ensures that the history cannot be changed or forged by admins or hackers. Should anybody change the log entries, the hashing of the modified block will have a conflicting hash, making the change detectable upon running a log audit.

Events Audited More than thirty events are monitored in the Audit Logs, giving the System Admin a full profile of events:

- Requesting, challenging, looking up, or deleting a certificate session or sub-session
- Creating, deleting, renaming, adding, or removing a directory entry
- Creating, deleting, or registering a proxy for virtual machines
- Attaching, detaching, creating, or deleting a virtual drive

  - and adding, retrieving, or deleting a key to a virtual drive
- Creating or deleting a file
  - and adding, retrieving, or deleting a key to a file
  - and reading or writing a chunk of the file
- Creating, deleting, or modifying members and permissions of a group Adding, deleting, or modifying permissions of a user