January 30, 2017

Memory enclaves with software configuration information

Uday Savagaonkar

Memory enclaves with software configuration information

ABSTRACT

Software guard extensions (SGX) allow an application to instantiate within memory a protected container, referred to as an enclave. An enclave is an area in the address space of an application that provides confidentiality and integrity even in presence of software of higher privilege, even if such software is malicious. The protection is achieved by restricting non-enclave accesses to code/data resident in the enclave, and by enforcing execution integrity for the enclave. An enclave has an identity, which comprises, for example, a hash of the code resident in the enclave, hash of a key with which the enclave was signed, a product version number and product category assigned by the software vendor, a hardware configuration of the enclave, etc. Within SGX, there are hardware-based mechanisms to attest to the identity of an enclave. There are also mechanisms to derive, using software and hardware, keys tied to a portion of the identity of the enclave. However, at present, there is no provision for enclave software to record the identity of its own configuration, or to seal secrets based on such recorded values. This disclosure describes mechanisms that allow enclave software to record and seal its configuration identity.

KEYWORDS

- Trusted computing
- Secure remote computing
- Software guard extensions
- Enclave

BACKGROUND

Cloud computing services enable a customer to provide software code to be executed on a computer provided by the cloud computing service provider. A customer may need to verify that such code and customer data is kept off-limits, e.g., from other customers that use the cloud computing service, the cloud computing service provider itself, etc. aside from access as explicitly authorized by the customer.

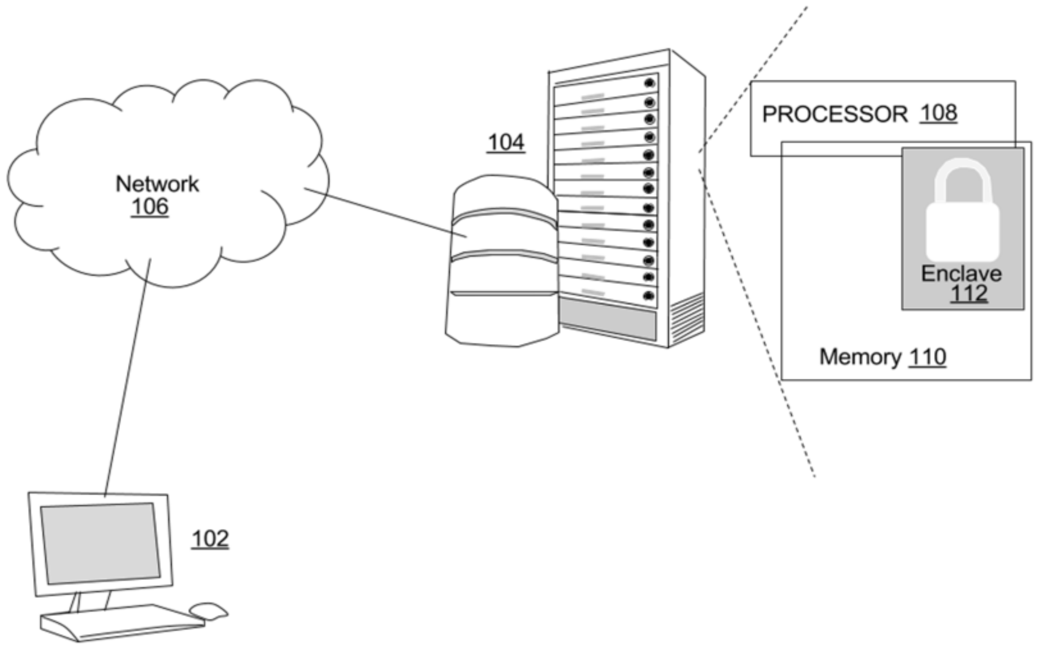


Fig. 1: An enclave within an address space of an application loaded by a customer on a remote server

Secure remote computing is the problem of executing software on a remote computer, e.g., a computer that is owned and operated by an untrusted party. Fig. 1 illustrates an example of secure remote computing. A customer connects using a client computer (102) to a remote computer (104) via a network (106) and uploads data and code to the remote computer. The remote computer may be untrusted, and the customer may want to verify that the uploaded data and code are secure, and used only in accordance with explicit authorization. Software guard extensions (SGX) solve this problem of secure remote computing by leveraging trusted hardware in the remote computer. The trusted hardware may comprise a processor (108) and memory (110). Within memory of the remote computer is established a secure container, known as an enclave (112). A customer of the remote computation service uploads code and data into the secure container. The trusted hardware protects the confidentiality and integrity of such code and data during computation.

An enclave is a restricted access execution environment. Access to an enclave's memory is restricted to software resident within the enclave and is prevented to software not resident in the enclave. Even software running at higher privilege levels, including malware, is not permitted access to the enclave. The enclave remains protected and provides confidentiality and integrity guarantees, even when, for example, the BIOS, Virtual Machine Manager (VMM) or hypervisor, Operating System (OS) or kernel, drivers, etc. are compromised. Even adversaries that possess full execution control over the remote computer are prevented from accessing code and data that are in the enclave, or tampering with execution of the enclave.

An enclave validates its integrity by using hardware-based mechanisms to respond to attestation challenges. Attestation proves to remote software that it is communicating with a specific piece of software running in a secure container hosted by trusted hardware. The proof is a cryptographic signature that verifies the identity of the contents of the secure container, which may comprise hash of the contents of the secure container, or a signature over the contents of the container, or some combination thereof. When the identity of the contents of an enclave, provided in response to a challenge, does not match an expected response, a customer of the remote-computation service can elect, for example, to not load further data or code into the enclave, or take other actions. Conversely, verification of the identity provides a high degree of certitude that the enclave is trustworthy.

For example, a customer that utilizes a cloud service for data analysis operations may desire to ensure that access to the execution code as well as data be restricted from other customers and the cloud service provider itself. Further, customers that utilize cloud services may also verify that a correct version (e.g., a verified untampered version) of execution code was applied in the processing of the data, e.g., to guarantee that output of the data analysis is reliable.

An enclave is managed via a collection of data structures. For example, the identity of an enclave is defined within a data structure known as the SGX Enclave Control Structure (SECS). The SECS has several fields that keep a track of the identity, e.g., a MISCSELECT field, an ATTRIBUTES field, an MRENCLAVE field, an MRSIGNER field, an ISVPRODID field, an ISVSVN field, etc. However, at present, fields that pertain to the identity do not record the configuration of the software running inside an enclave. Such fields are also not usable to provision or seal secrets based on the recorded value. Lack of such fields precludes a customer from launching an enclave written/signed by one vendor, and customizing that enclave to their own needs based on security-sensitive configuration information (e.g., public key of the customer's smart card).

DESCRIPTION

This disclosure describes techniques that enable recording of software configuration within the enclave identity, thereby making software configuration part of the enclave identity. For example, two fields are introduced that pertain to software configuration. The two fields may be inserted in a data structure that manages the enclave, e.g., the SECS data structure.

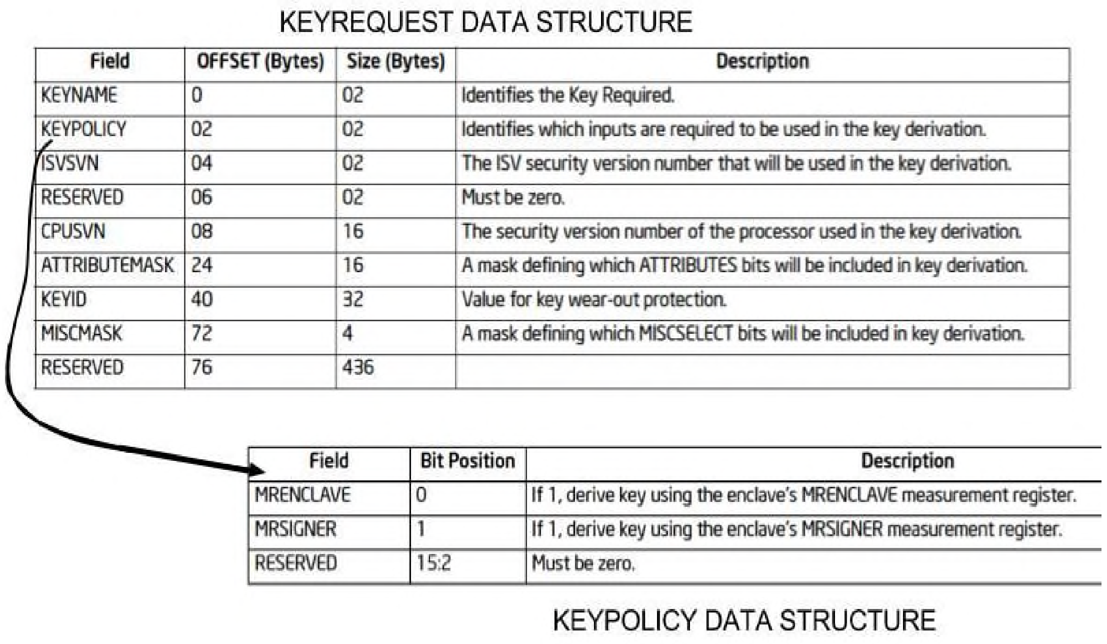
The first of the two fields is referred to as SWCFGNONMONOTONIC, which may, for example, span 256 bits. Any entity that requires checking of software configuration can perform a comparison with SWCFGNONMONOTONIC to verify an exact match with expectation. For example, software executing on a customer's on premise computers, or software executing in other enclaves (e.g., on a computer of the remote-computation service provider) can perform such verification.

The second of the two fields is referred to as SWCFGMONOTONIC, which may, for example, be 32 bits wide. The field SWCFGMONOTONIC is greater than or equal to the expected value. Any entity that requires checking of software configuration can perform a comparison with SWCFGMONOTONIC to verify that SWCFGMONOTONIC is greater than or equal to an expected value.

Access to the above two fields may be enabled by extending the instruction set of a processor that provides the enclave to enable software code write to these two fields in a write-once fashion. The write-once semantics of these fields ensure that subsequent attempts to write to these fields cause a failure, either by way of error or exception. For example, when software is initially loaded to an enclave, a configuration is provided as an input to the enclave. The fields as described herein are programmed (e.g., assigned values) based on the configuration prior to the software execution. Since the fields are write once, a bad software configuration is not able to rewrite the fields in the enclave, e.g., with incorrect values. For example, if such software attempts to load a dynamic library to the enclave, e.g., a library that attempts to change values of the fields, rewriting the fields to hide the fact that a bad library was loaded is prevented.

Within the SGX instructions, there is a leaf function ENCLU[EREPORT]. Leaf functions are those that don't call other functions. The leaf function ENCLU[EREPORT] can only be executed inside an enclave and creates a cryptographic report of the enclave. Per techniques of this disclosure, the ENCLU[EREPORT] leaf function is extended to include in the hardware-generated report contents of the fields SWCFGMONOTONIC and SWCFGNONMONOTONIC.

Within the SGX instructions, there is also a leaf function ENCLU[EGETKEY], which returns a secret key from the processor-specific key hierarchy. ENCLU[EGETKEY] derives keys using a value unique to a processor in order to create a specific key. ENCLU[EGETKEY] can only be executed inside an enclave. An input to the ENCLU[EGETKEY] function is a data structure known as KEYREQUEST, which is used to select from one of the processor-specific keys, and set additional parameters required in the derivation of the selected key. Within the KEYREQUEST data structure is another data structure known as KEYPOLICY. Fig. 2 illustrates the data structures KEYREQUEST and KEYPOLICY.



KEYREQUEST DATA STRUCTURE

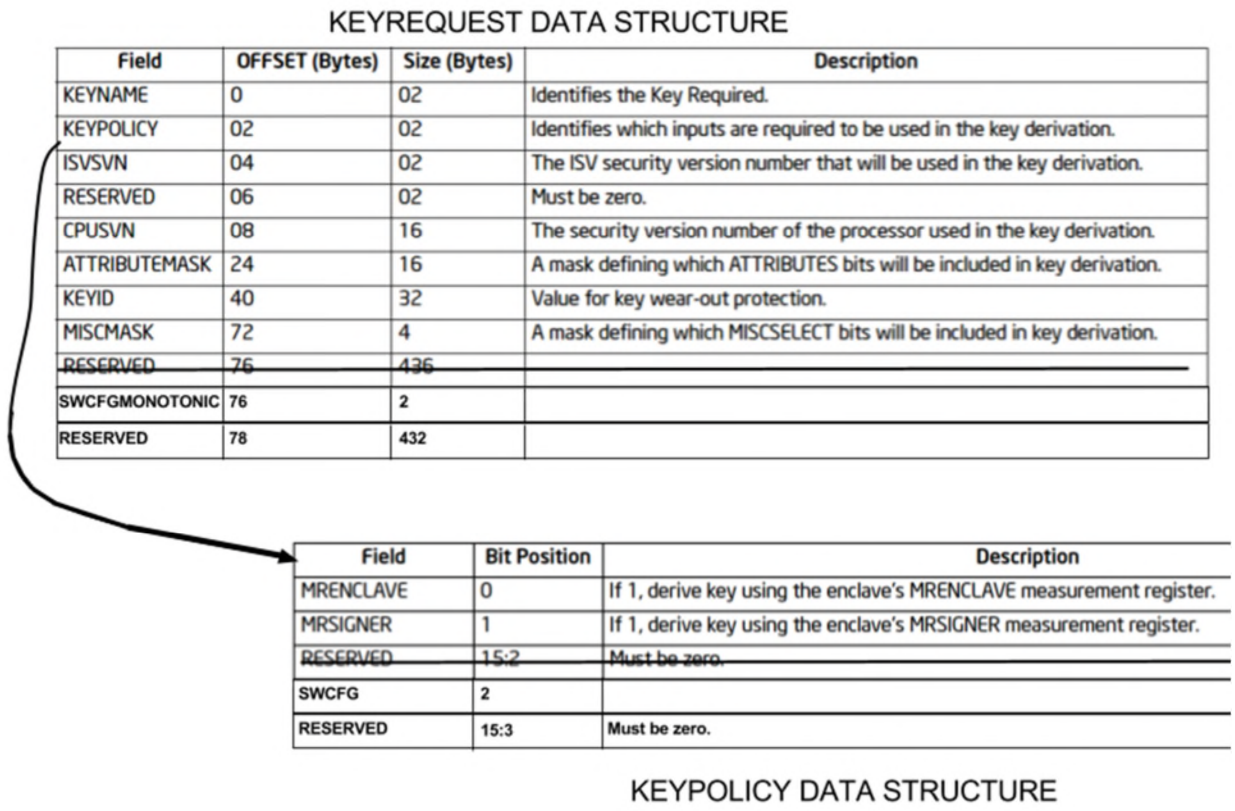
Field	OFFSET (Bytes)	Size (Bytes)	Description
KEYNAME	0	02	Identifies the Key Required.
KEYPOLICY	02	02	Identifies which inputs are required to be used in the key derivation.
ISVSVN	04	02	The ISV security version number that will be used in the key derivation.
RESERVED	06	02	Must be zero.
CPUSVN	08	16	The security version number of the processor used in the key derivation.
ATTRIBUTEMASK	24	16	A mask defining which ATTRIBUTES bits will be included in key derivation.
KEYID	40	32	Value for key wear-out protection.
MISCMASK	72	4	A mask defining which MISCSELECT bits will be included in key derivation.
RESERVED	76	436	

Field	Bit Position	Description
MRENCLAVE	0	If 1, derive key using the enclave's MRENCLAVE measurement register.
MRSIGNER	1	If 1, derive key using the enclave's MRSIGNER measurement register.
RESERVED	15:2	Must be zero.

KEYPOLICY DATA STRUCTURE

Fig. 2: Fields of the KEYREQUEST and KEYPOLICY data structures, and their mutual relationship (from [1])

Per techniques of this disclosure, the KEYREQUEST data structure is extended to add a SWCFGMONOTONIC field, which may be, for example, 32 bits wide. Further, per techniques disclosed herein, the KEYPOLICY data structure is extended to add a SWCFG bit. An example illustration of the resulting KEYREQUEST and KEYPOLICY data structures is shown in Fig. 3.



KEYREQUEST DATA STRUCTURE

Field	OFFSET (Bytes)	Size (Bytes)	Description
KEYNAME	0	02	Identifies the Key Required.
KEYPOLICY	02	02	Identifies which inputs are required to be used in the key derivation.
ISVSVN	04	02	The ISV security version number that will be used in the key derivation.
RESERVED	06	02	Must be zero.
CPUSVN	08	16	The security version number of the processor used in the key derivation.
ATTRIBUTEMASK	24	16	A mask defining which ATTRIBUTES bits will be included in key derivation.
KEYID	40	32	Value for key wear-out protection.
MISCMASK	72	4	A mask defining which MISCSELECT bits will be included in key derivation.
~~RESERVED~~	~~76~~	~~436~~	
SWCFGMONOTONIC	76	2	
RESERVED	78	432	

Field	Bit Position	Description
MRENCLAVE	0	If 1, derive key using the enclave's MRENCLAVE measurement register.
MRSIGNER	1	If 1, derive key using the enclave's MRSIGNER measurement register.
~~RESERVED~~	~~15:2~~	~~Must be zero.~~
SWCFG	2	
RESERVED	15:3	Must be zero.

KEYPOLICY DATA STRUCTURE

Fig. 3: An example modification of the KEYREQUEST and KEYPOLICY data structures, per techniques of this disclosure.

Once input to the ENCLU[EGETKEY] function is modified, for example, as shown in Fig. 3, the behavior of the ENCLU[EGETKEY] leaf function is defined in accordance with the following rules:

1. If the KEYPOLICY→SWCFG bit is set, then two enclaves with different values of SECS→SWCFGNONMONOTONIC get cryptographically independent keys, irrespective of other portions of identities of the two enclaves. That is, if the KEYPOLICY→SWCFG bit is set, then the key returned by the ENCLU[EGETKEY] leaf function is bound to the software configuration portion of the enclave identity.

2. Two ENCLU[EGETKEY] invocations, either from the same enclave or from different enclaves, with two different values of KEYREQUEST→SWCFGMONOTONIC result in cryptographically independent keys, irrespective of the other portions of the identity.
3. An ENCLU[EGETKEY] invocation succeeds if and only if the KEYREQUEST→SWCFGMONOTONIC has a value that is less than or equal to that of SECS→SWCFGMONOTONIC.

These extensions are similar to the functionality of the ENCLU[EGETKEY] leaf with regard to MRSIGNER, ISVPRODID, ISVSVN, etc. portions of enclave identity. Similar extensions are made to enclaves that attest to the identity of another enclave.

Modifications to the SGX instruction set, as described above, enable software running inside an enclave to record the identity of its configuration, that is not otherwise captured by existing enclave identity semantics. Further, per techniques described herein, secrets may be sealed and/or provisioned based on the recorded value.

Per this disclosure, the hardware does not assign any specific meaning to the value of SWCFGNONMONOTONIC. It is completely up to the enclave software and the remote customer software to assign a meaning to SWCFGNONMONOTONIC. Also, the hardware is configured only for the monotonic nature of the SWCFGMONOTONIC field to be valid. The hardware configuration does not assign any specific meaning to SWCFGMONOTONIC, other than for the monotonic relationship between the two fields to be valid.

<u>CONCLUSION</u>

This disclosure describes techniques to record the configuration or identity of software running within an enclave, which is a secure container within the memory of a computer. For example, two fields are added to a data structure that manages the enclave. The first of the two fields serves to verify an exact match of software configuration with expectation. The second of the two fields serves to verify that the field is greater than or equal to expectation. Modifications in the data structure and leaf functions are proposed to support the functionalities described herein. Per techniques of this disclosure, software executing inside an enclave can record the identity of its configuration, and provision/seal secrets based on the recorded value.

REFERENCES

[1] "Intel 64 and IA32 Architectures Software Developer's Manual," *Volume. 3D: System Programming Guide Part 4*, Sep. 2016.