(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
5 February 2015 (05.02.2015)

WIPO | PCT

(10) International Publication Number
WO 2015/016994 A1

(51) International Patent Classification:
G06F 21/72 (2013.01) *G06F 21/76* (2013.01)

(21) International Application Number:
PCT/US2014/036675

(22) International Filing Date:
2 May 2014 (02.05.2014)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
13/954,487 30 July 2013 (30.07.2013) US

(71) Applicant: **BATTELLE MEMORIAL INSTITUTE** [US/US]; P.O. Box 999, K1-53, Richland, Washington 99352 (US).

(72) Inventors: **GRISWOLD, Richard L.**; 5503 Kalakaua Court, West Richland, Washington 99353 (US). **NICKLESS, William K.**; 1761 George Washington Way, Suite 171, Richland, Washington 99354 (US). **CONRAD, Ryan C.**; 2360 Morgan Court, West Richland, Washington 99353 (US).

(74) Agent: **GOKCEK, A.J.**; P.O. Box 999, K1-53, Richland, Washington 99352 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

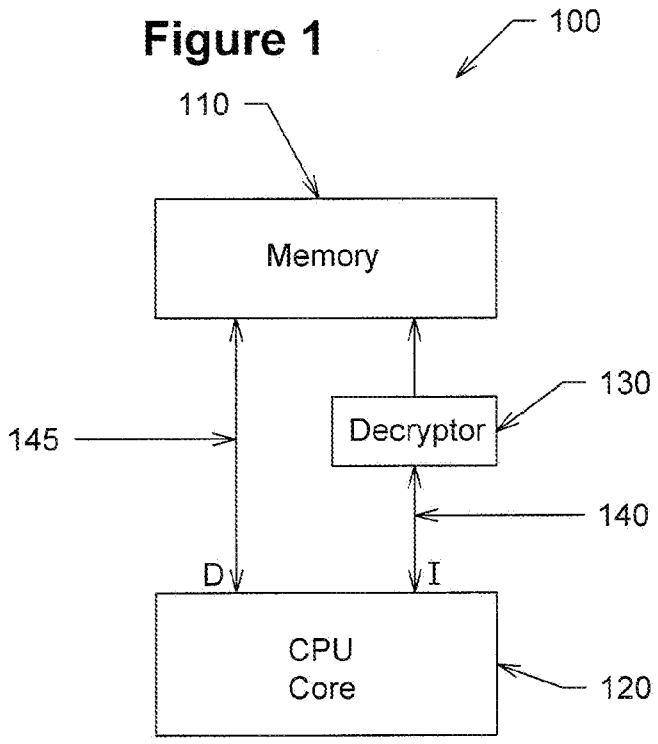
(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ,

[Continued on next page]

(54) Title: SYSTEM FOR PROCESSING AN ENCRYPTED INSTRUCTION STREAM IN HARDWARE

(57) Abstract: A system and method of processing an encrypted instruction stream in hardware is disclosed. Main memory stores the encrypted instruction stream and unencrypted data. A central processing unit (CPU) is operatively coupled to the main memory. A decryptor is operatively coupled to the main memory and located within the CPU. The decryptor decrypts the encrypted instruction stream upon receipt of an instruction fetch signal from a CPU core. Unencrypted data is passed through to the CPU core without decryption upon receipt of a data fetch signal.

Figure 1



WO 2015/016994 A1

TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

SYSTEM FOR PROCESSING AN ENCRYPTED INSTRUCTION STREAM IN HARDWARE

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

[0001] This invention claims priority to U.S. Patent Application Number 13/954,487, filed July 30, 2013, entitled *SYSTEM FOR PROCESSING AN ENCRYPTED INSTRUCTION STREAM IN HARDWARE.*

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

[0002] The invention was made with Government support under Contract DE-AC05-76RLO1830, awarded by the U.S. Department of Energy. The Government has certain rights in the invention.

TECHNICAL FIELD

[0003] This invention relates to cyber security. More specifically, this invention relates to instruction stream randomization by providing support in hardware for executing encrypted code running in a central processing unit (CPU).

BACKGROUND OF THE INVENTION

[0004] Successful cyber attacks often leverage the fact that an instruction set architecture of a target system is well known. Given knowledge of the instruction set architecture, attackers can prepare malicious software, knowing with high confidence that it will run once introduced into the target system via code-injection attacks or other attack vectors.

[0005] Instruction stream randomization (ISR) seeks to thwart these attacks by creating unique, dynamic system architectures, thus denying attackers the asymmetric advantage of a well-known target architecture by forcing them to expend considerable resources for each system they wish to compromise. However, previous ISR research has been hampered by the need for hardware emulators to implement the necessary changes to the CPU.

SUMMARY OF THE INVENTION

[0006] In accordance with one embodiment of the present invention, a system for processing an encrypted instruction stream in hardware is disclosed. The system includes a main memory for storing the encrypted instruction stream and unencrypted data. The system also includes a CPU operatively coupled to the main memory via a unified instruction and data bus. The system further includes a decryptor coupled to the unified instruction and data bus. The decryptor decrypts the encrypted instruction stream upon receipt of an instruction fetch signal from a CPU core, and the decryptor passes the unencrypted data through without decryption upon receipt of a data fetch signal from the CPU core.

[0007] The system may further comprise a cache for receiving the decrypted instruction stream from the decryptor, wherein the decryptor is coupled between the cache and the main memory. In one embodiment, the cache is not directly accessible from instructions executing on the CPU.

[0008] The system may further comprise a memory controller coupled between the cache and the CPU core. The memory controller receives the decrypted stream and the unencrypted data.

[0009] The system may further comprise a boot controller for initializing the CPU to start executing the encrypted instruction stream immediately without requiring an unencrypted software boot strapping routine.

[0010] In one embodiment, during the initialization the CPU reads a cryptographic key from dedicated storage and a nonce value from a dedicated address in the instruction stream. A key used by the decryptor is derived from, but not limited to, at least one of the following: the cryptographic key, the nonce, or a CPU counter. The cryptographic key can be contained within an internal register of the decryptor.

[0011] In one embodiment, the key is derived using an Advanced Encryption Standard (AES) algorithm with a 128-bit key length.

[0012] In one embodiment, the nonce is located at the beginning of the instruction stream. The nonce can be generated anew each time the instruction stream is encrypted.

[0013] In one embodiment, the instruction stream is periodically re-encrypted at intervals during operation of the CPU.

[0014] The CPU is, but not limited to, a MIPS CPU, an ARM-based CPU, or an x86 CPU, and may be implemented in a field-programmable gate array (FPGA). Alternatively, the CPU may be implemented in an application-specific integrated circuit (ASIC).

[0015] In one embodiment, the decryptor uses an AES algorithm in counter mode (AES-CTR) with a 128-bit key length. Other encryption standards and key lengths, such as a 196-bit key length or 256-bit key length, may be used by the decryptor.

[0016] The main memory is, but not limited to, a random-access memory (RAM). The RAM is, but not limited to, a synchronous dynamic RAM (SDRAM).

[0017] The decryptor can utilize a checksum or a hash value to detect an improperly decrypted instruction stream.

[0018] The system can re-initialize the CPU to a predefined state when the improperly decrypted instruction stream is detected. In one embodiment, the system sets a CPU program counter to a non-sequential value when the improperly decrypted instruction stream is detected.

[0019] In another embodiment of the present invention, a system for processing an encrypted instruction stream in hardware is disclosed. The system includes a main memory for storing the encrypted instruction stream and unencrypted data. The system also includes a CPU operatively coupled to the main memory via a separate instruction bus and data bus. The system further includes a decryptor coupled to the instruction bus but not the data bus. The decryptor decrypts the encrypted instruction stream upon receipt of an instruction via the instruction bus.

[0020] In another embodiment of the present invention, a system for processing an encrypted instruction stream in hardware is disclosed. The system includes a main memory for storing the encrypted instruction stream and unencrypted data. The system also includes a CPU operatively coupled to the main memory. The system further includes a decryptor operatively coupled to the main memory and located within the CPU. The decryptor decrypts the encrypted instruction stream upon receipt of an instruction fetch signal from a CPU core. Unencrypted data is passed through to the CPU core without decryption upon receipt of a data fetch signal.

[0021] In another embodiment of the present invention, a method of initializing a decryptor is disclosed. The method includes pausing a CPU, wherein a program counter does not increment, while a boot controller performs the following: reading a nonce value from a first predetermined location; storing the nonce value in a first hardware register of the decryptor; reading a cryptographic key from a second predetermined location; storing the cryptographic key in a second hardware register of the decryptor; forming an initial counter value from the nonce value; and sending the initial counter value to the decryptor, wherein the CPU resumes operations after the decryptor initializes.

BRIEF DESCRIPTION OF THE DRAWINGS

[0022] Figure 1 illustrates a block diagram of a system for processing an encrypted instruction stream in hardware displaying a CPU operatively coupled to the main memory via a separate instruction bus and data bus, with a decryptor coupled to the instruction bus but not the data bus, in accordance with one embodiment of the present invention.

[0023] Figure 2 illustrates a block diagram of a system for processing an encrypted instruction stream in hardware displaying a CPU operatively coupled to the main memory via a decryptor that is coupled to a unified instruction and data bus, in accordance with one embodiment of the present invention.

[0024] Figure 3 illustrates a block diagram for processing an encrypted instruction stream, in accordance with one embodiment of the present invention.

[0025] Figure 4 illustrates a counter value that is 128 bits in length, with a 32-bit address padded with 32 bits of zero.

[0026] Figure 5 illustrates a block diagram of the encryption and decryption process, with the encryption performed off line, in accordance with one embodiment of the present invention.

[0027] Figure 6 is a graph of the test results generated by applying random instruction streams to mimic improperly encrypted instruction streams that were fed into a process core using the architecture as depicted in Figure 3. The test results show the percentage of improperly encrypted instruction streams that halted after n instructions.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0028] The Present Invention includes systems and methods of processing an encrypted instruction stream in hardware are disclosed. These systems and methods prevent the

successful execution of code-injection attacks and, more broadly, the successful execution of any malicious or unauthorized binary code on a system.

[0029] In one embodiment, the Present Invention also replaces the cryptographically insecure methods with a cryptographically secure cipher, turning ISR into "instruction stream encryption". In another embodiment, the Present Invention also does away with slow and vulnerable software infrastructures in favor of placing all of the components necessary to support instruction stream encryption directly into hardware. This provides much faster execution times and reduced attack surface, which increases system security.

[0030] The Present Invention can protect all code that executes on the system from the very first instruction. In one embodiment, by having no unencrypted instructions the Present Invention eliminates windows of opportunity for hackers. The Present Invention also does not require a software infrastructure or helper modules to support the execution of the encrypted instruction stream.

[0031] In one embodiment, an implementation of ISR in a soft-core CPU capable of directly executing an Advanced Encryption Standard (AES) encrypted instruction stream is disclosed which does not require an emulation layer or additional software components. This provides a direct avenue for higher performance implementations in ASICs and custom semiconductor fabrications. Implementation of this instruction stream encryption complements existing security infrastructure and provides strong protection for high assurance environments where there is a high probability of compromise. Design goals for instruction stream encryption include, but are not limited to, high performance, cryptographically secure encryption, and self-containment.

[0032] In some embodiments, the implementation is in hardware rather than relying on a hardware emulation layer. The CPU may be implemented in a FPGA, which provides a

direct path to higher performance implementations in ASICs or custom semiconductor fabrications.

[0033] AES in counter mode (AES-CTR) may be used for the encryption algorithm. Implemented properly, AES-CTR provides high resilience against cryptanalysis and requires only enough memory to store the encryption key and counter value. With a 128-bit key length, this comes to 256 bits of storage for the key and counter value.

[0034] With a self-contained implementation, the CPU does not have to rely on other software components, such as encrypting loaders, to initialize or manage the ISR subsystem. Removing the need for such software components reduces the attack surface, because software no longer needs access to the encryption key. By placing the encryption key outside of software access, an attacker cannot coerce the system into divulging the encryption key.

[0035] Figure 1 illustrates a block diagram of a system 100 for processing an encrypted instruction stream in hardware displaying a CPU 120 operatively coupled to the main memory 110 via a separate instruction bus 140 and data bus 145, with a decryptor130 coupled to the instruction bus 140 but not the data bus 145, in accordance with one embodiment of the present invention. The main memory 110 stores an encrypted instruction stream and unencrypted data. The decryptor 130 decrypts the encrypted instruction stream upon receipt of an instruction via the instruction bus 140.

[0036] Figure 2 illustrates a block diagram of a system 200 for processing an encrypted instruction stream in hardware displaying a CPU 220 operatively coupled to the main memory 210 via a decryptor 230 that is coupled to a unified instruction and data bus 250, in accordance with one embodiment of the present invention. The decryptor 230 may receive instructions and data from the main memory 210 via separate instruction and data buses 240 and 245, respectively.

[0037] In one embodiment, as mentioned above, the Present Invention uses AES in counter mode for the encryption algorithm. An element that adds to the robustness of AES-CTR is the use of a nonce, or "number used once", wherein a counter value with the same encryption key may be reused without compromising security. The counter value may be generated by concatenating a 64-bit nonce with the address of the start of each block of instructions as depicted in Figure 4. Because the counter value is, in this example, 128 bits in length, the 32-bit address is padded with 32 bits of zeros. The 64-bit nonce is randomly generated each time the software is encrypted for the device, thus ensuring that the same counter value is never used twice with the same key. The counter value is encrypted with each device's unique key to create the cipher block to encrypt each block of instructions. A 128-bit key is used, in this embodiment, so each key stream block holds four fixed-sized 32-bit instructions.

[0038] In one embodiment, since software access to the system is not allowed, all encryption must be done offline as depicted in Figure 5. This prevents attackers from creating properly encrypted code on a device that is using ISR.

[0039] In one embodiment, the Present Invention uses the same key for encryption and decryption, so the encryption key is stored on the device that uses ISR. Key storage and key register are not software addressable. During initialization, the processor reads the encryption key from dedicated storage and the nonce from, as one example, the first eight bytes of memory. It uses the nonce and the program counter to create the counter value as described previously. The nonce may be read from other locations.

[0040] One effect of disallowing software access to the system is that software has no mechanism for switching the CPU between encrypted and unencrypted mode. Instead, the CPU always operates in encrypted mode, starting with the first instruction it executes. This

prevents an attacker from forcing the CPU into unencrypted mode, since the CPU is incapable of operating in unencrypted mode. The system provides a mechanism or the CPU to get the three components needed to decrypt the instructions: the encryption key, the nonce, and the current program counter value, without relying on software support. The encryption key may be hard-coded into the soft-core processor image, but it could read from any dedicated non-volatile storage. The nonce does not need to be protected as carefully as the encryption key, so it can be stored at a known location in memory. In one embodiment, the nonce is stored at address 0, before the start of bootloader code.

[0041] Figure 3 illustrates a block diagram of a system for processing an encrypted instruction stream, in accordance with one embodiment of the present invention. In this embodiment, the CPU of the system requires that no unencrypted instructions are executed, and that no user data is decrypted. To ensure this, the decryptor (or decryptor interface) is placed between the CPU and the main memory, SDRAM. In the embodiment of Figure 3, the SDRAM stores the encrypted instruction stream and unencrypted data. The CPU controller, which includes registers, multiplexors, a program counter, and an arithmetic logic unit, is operatively coupled to the main memory via a separate instruction bus and data bus. The decryptor is coupled to the instruction bus but not the data bus. The Ethernet Controller has no direct access to the CPU and other peripherals do not have the data width to execute instructions.

[0042] In one embodiment, the decryptor reads a key from an internal register and builds subkeys calculated for each round of encryption. The key is inaccessible by the user. The 64-bit nonce is read and stored as a fixed upper half of the counter value. After these startup routines, the CPU is enabled and the decryptor examines all data to determine if decryption is necessary. The decryptor may decrypt the data if it is flagged as an instruction rather user data and/or it is located in the instruction address space of memory. These conditions prevent

an attacker from executing unencrypted instructions from user data address space and catch any exceptions in instruction address space.

[0043] To mitigate any decreased performance from run-time decryption, the system of Figure 3 may include, but is not limited to, several additions: an unencrypted instruction cache, a cipher block cache, and decryption clock multiplication. A direct mapped cache stores unencrypted instructions, thus reducing the total number of decryptions necessary. The cache is only addressable by the memory controller, which is not software/user accessible. The decrypted cipher block cache allows the CPU to bypass the decryption phase for sequential instructions. In one embodiment, for each decryption processed, four words are decrypted, thus reducing the overall decryption time by approximately 75%. Decryption clock multiplication reduces decryption latency. Since propagation delay through the decryption core is less that of the CPU, the decryption clock can be run at higher frequencies than the CPU clock.

[0044] The system may optionally use a dedicated hardware boot controller. An implementation with limited local static RAM/PROM may utilize a boot controller to allow a larger bootloader and simplify bootloader addressing. A boot controller that initializes main memory and copies the encrypted bootloader code into the main memory was designed. The CPU then begins executing the bootloader code, at which point a user can load more encrypted code and unencrypted user data into the main memory via the UART interface. This embodiment preserves the security features of the system and allows running from a single memory peripheral.

Experimental Section

[0045] The following examples serve to illustrate embodiments and aspects of the present invention and are not to be construed as limiting the scope thereof.

[0046] In order to provide proof of concept, the ISR CPU system of Figure 3 was implemented on a Spartan 3E Starter Board using the open-source Plasma soft-core CPU. This development board contains a low-grade Xilinx FPGA and basic user peripherals supported by Plasma such as UART, SRAM, DDR, and Ethernet. The CPU was interfaced with a basic decryption core, the Avalon AES ECB-core, by adding a small finite state machine (FSM) to coordinate memory fetches, cache checks/misses, and data decryption. Performance optimizations and a hardware boot controller were added to ensure seamless startup and normal operations. The Plasma system includes the source code for the processor as well as an emulator and a small real-time operating system (RTOS) with a network stack and web server.

[0047] A simple benchmark was chosen for performance testing. The Plasma RTOS comes with a HTTP server, which was configured to serve the same image in three different formats: as a 41,733 byte GIF, an 11,088 byte JPEG, and a 3,444 byte PNG. The test program downloads the image 100 times in each format, while measuring the elapsed time. This team ensures that the decryptor is exercised and its performance factors into the measurements, since the 4kB decrypted instruction cache is not large enough to hold all of the code used in handling the network traffic and HTTP requests. Elapsed time for running on the processor without encryption was 308.64s, while elapsed time with encryption enabled was 323.91s, or an increase of 4.95% in run time.

[0048] A simulation was run to determine how many instructions, on average, the processor would execute from an improperly encrypted instruction stream before halting. Since AES-CTR does not validate the integrity of the encrypted instruction stream, the

encryption engine passes the results of the decryption to the processor core to execute regardless of whether the instruction stream was properly encrypted. However, when the incoming instruction stream is not properly encrypted, the resulting instruction stream will contain invalid instructions or memory accesses. 500,000 random instruction streams were generated to mimic improperly encrypted instruction streams, and fed them into the processor core. The test results are given in Table 1 below and shown graphically in Figure 6. Almost 64% of the time the system halted on the first instruction. Over 99% of the time, the CPU encountered an illegal instruction or other malformed instruction that caused an exception or interrupt within seven instructions. In no instance did it run for more than 18 instructions without experiencing an exception or interrupt, unless the invalid instruction stream placed the system in a hard loop.

Table 1

Instructions Before Halting	Individual Tests		Cumulative	
	Count	Percent	Count	Percent
1	318488	63.70%	318488	63.70%
2	34138	6.83%	352626	70.53%
3	83243	16.65%	435869	87.17%
4	17209	3.44%	453078	90.62%
5	24339	4.87%	477417	95.48%
6	10511	2.10%	487928	97.59%
7	8473	1.69%	496401	99.28%
8	1301	0.26%	497702	99.54%
9	1140	0.23%	498842	99.77%
10	782	0.16%	499624	99.92%
11	43	0.01%	499667	99.93%
12	278	0.06%	499945	99.99%
13	8	0.00%	499953	99.99%
14	5	0.00%	499958	99.99%
15	3	0.00%	499961	99.99%
16	1	0.00%	499962	99.99%
17	1	0.00%	499963	99.99%
18	1	0.00%	499964	99.99%
Hard loop	36	0.01%	500000	100.00%

[0049] As this example showed, the system can efficiently execute a fully encrypted instruction stream and successfully block improperly encrypted code.

[0050] As discussed above, the AES-CTR algorithm does not provide verification of the decrypted results. Instead, the system relies on the statistical probability that the improperly encrypted instruction stream will contain an invalid instruction or memory access. The results above show that 99% of the time, the system will halt within seven instructions. However, the system is not guaranteed to encounter a malformed or illegal instruction that causes an exception or interrupt, and about 0.01% of the time it will instead go into a hard loop.

[0051] In one embodiment, switching to another encryption algorithm which provides both integrity and confidentiality, such as AES in Galois Counter Mode (AES-GCM), would

allow the hardware to detect an improperly encrypted instruction prior to execution. The system could then distinguish between illegal instruction and addressing errors, and improperly encrypted instruction stream errors and then respond in a controlled, deterministic manner. For example, if an incorrect encryption key is detected, such as via a decryption error message, the system can take one or more protective actions. These protective actions include, but are not limited to, the following: resetting to a known "good" code, raising an interrupt, alerting operators, or dropping the instruction frame.

[0052] The embodiments described above have broad uses such as, but not limited to, the energy sector, critical infrastructure, security, and areas that involve network enabled embedded devices. In the embedded systems space, one specific application would be for smart grid meters which allow access to the electrical grid infrastructure, and in some cases devices in customers' homes, from a computer network.

[0053] The present invention has been described in terms of specific embodiments incorporating details to facilitate the understanding of the principles of construction and operation of the invention. As such, references herein to specific embodiments and details thereof are not intended to limit the scope of the claims appended hereto. It will be apparent to those skilled in the art that modifications can be made in the embodiments chosen for illustration without departing from the spirit and scope of the invention.

CLAIMS

We claim:

1. A system for processing an encrypted instruction stream in hardware comprising:
    a. a main memory for storing the encrypted instruction stream and unencrypted data;
    b. a central processing unit (CPU) operatively coupled to the main memory via a unified instruction and data bus; and
    c. a decryptor coupled to the unified instruction and data bus, wherein the decryptor decrypts the encrypted instruction stream upon receipt of an instruction fetch signal from a CPU core and wherein the decryptor passes the unencrypted data through without decryption upon receipt of a data fetch signal from the CPU core.
2. The system of Claim 1 further comprising a cache for receiving the decrypted instruction stream from the decryptor, wherein the decryptor is coupled between the cache and the main memory.
3. The system of Claim 2 wherein the cache is not directly accessible from instructions executing on the CPU.
4. The system of Claim 2 further comprising a memory controller coupled between the cache and the CPU core, wherein the memory controller receives the decrypted stream and the unencrypted data.
5. The system of Claim 1 further comprising a boot controller for initializing the CPU to start executing the encrypted instruction stream immediately without requiring an unencrypted software boot strapping routing.
6. The system of Claim 5 wherein during the initialization the CPU reads a cryptographic key from dedicated storage and a nonce value from a dedicated address in the instruction stream.

7. The system of Claim 6 wherein a key used by the decryptor is derived from at least one of the following: the cryptographic key, the nonce, and a CPU program counter.
8. The system of Claim 6 wherein the nonce is located at the beginning of the instruction stream.
9. The system of Claim 8 wherein the nonce is generated anew each time the instruction stream is encrypted.
10. The system of Claim 6 wherein the cryptographic key is contained within an internal register of the decryptor.
11. The system of Claim 10 wherein the internal register is not software or user accessible.
12. The system of Claim 1 wherein the instruction stream is periodically re-encrypted at intervals during operation of the CPU.
13. The system of Claim 1 wherein the CPU is implemented in a field-programmable gate array (FPGA).
14. The system of Claim 1 wherein the CPU is at least one of the following: a MIPS CPU, an ARM-based CPU, and an x86 CPU.
15. The system of Claim 1 wherein the decryptor uses an Advanced Encryption Standard (AES) algorithm in counter mode (AES-CTR) with a 128-bit key length.
16. The system of Claim 1 wherein the main memory is a random-access memory.
17. The system of Claim 16 wherein the random-access memory is a synchronous dynamic random-access memory (SDRAM).
18. The system of Claim 1 wherein the decryptor utilizes a checksum or a hash value to detect an improperly decrypted instruction stream.
19. The system of Claim 18 wherein the system re-initializes the CPU to a predefined state when the improperly decrypted instruction stream is detected.

20. The system of Claim 18 wherein the system sets a CPU program counter to a non-sequential value when the improperly decrypted instruction stream is detected.

21. A system for processing an encrypted instruction stream in hardware comprising:

    a. a main memory for storing the encrypted instruction stream and unencrypted data;

    b. a central processing unit (CPU) operatively coupled to the main memory via a separate instruction bus and data bus; and

    c. a decryptor coupled to the instruction bus but not the data bus, wherein the decryptor decrypts the encrypted instruction stream upon receipt of an instruction via the instruction bus.

22. The system of Claim 21 further comprising a cache for receiving the decrypted instruction stream from the decryptor, wherein the decryptor is coupled between the cache and the main memory.

23. The system of Claim 22 wherein the cache is not directly accessible from instructions executing on the CPU.

24. The system of Claim 22 further comprising a memory controller coupled between the cache and the CPU core, wherein the memory controller receives the decrypted stream and the unencrypted data.

25. The system of Claim 21 further comprising a boot controller for initializing the CPU to start executing the encrypted instruction stream immediately without requiring an unencrypted software boot strapping routing.

26. The system of Claim 25 wherein during the initialization the CPU reads a cryptographic key from dedicated storage and a nonce value from a dedicated address in the instruction stream.

27. The system of Claim 26 wherein a key used by the decryptor is derived from at least one of the following: the cryptographic key, the nonce value, and a CPU program counter.

28. The system of Claim 26 wherein the nonce value is located at the beginning of the instruction stream.

29. The system of Claim 28 wherein the nonce value is generated anew each time the instruction stream is encrypted.

30. The system of Claim 26 wherein the cryptographic key is contained within an internal register of the decryptor.

31. The system of Claim 30 wherein the internal register is not software or user accessible.

32. The system of Claim 21 wherein the instruction stream is periodically re-encrypted at intervals during operation of the CPU.

33. The system of Claim 21 wherein the CPU is implemented in a field-programmable gate array (FPGA).

34. The system of Claim 21 wherein the CPU is at least one of the following: a MIPS CPU, an ARM-based CPU, and an x86 CPU.

35. The system of Claim 21 wherein the decryptor is a 128-bit Advanced Encryption Standard (AES) decryptor.

36. The system of Claim 21 wherein the main memory is a random-access memory.

37. The system of Claim 36 wherein the random-access memory is a synchronous dynamic random-access memory (SDRAM).

38. The system of Claim 21 wherein the decryptor utilizes a checksum or a hash value to detect an improperly decrypted instruction stream.

39. The system of Claim 38 wherein the system re-initializes the CPU to a predefined state when the improperly decrypted instruction stream is detected.

40. The system of Claim 38 wherein the system sets a CPU program counter to a non-sequential value when the improperly decrypted instruction is detected.

41. A system for processing an encrypted instruction stream in hardware, comprising:

a. a main memory for storing the encrypted instruction stream and unencrypted data;

b. a central processing unit (CPU) operatively coupled to the main memory; and

c. a decryptor operatively coupled to the main memory and located within the CPU, wherein the decryptor decrypts the encrypted instruction stream upon receipt of an instruction fetch signal from a CPU core, and wherein unencrypted data is passed through to the CPU core without decryption upon receipt of a data fetch signal.

42. The system of Claim 41 further comprising a unified instruction and data bus coupled between the decryptor and the CPU core, wherein the decryptor decrypts the encrypted instruction stream upon receipt of an instruction fetch signal from the CPU core and passes the unencrypted data through without decryption upon receipt of a data fetch signal from the CPU core.

43. The system of 41 further comprising a data bus coupled between the CPU core and the main memory and an instruction bus coupled between the CPU core and the main memory via the decryptor, the decryptor being coupled to the instruction bus but not the data bus, wherein the decryptor decrypts the encrypted instruction stream upon receipt of an instruction via the instruction bus.

44. A method of initializing a decryptor comprising: pausing a CPU, wherein a program counter does not increment, while a boot controller performs the following:

a. reading a nonce value from a first predetermined location;

b. storing the nonce value in a first hardware register of the decryptor;

c. reading a cryptographic key from a second predetermined location;

d. storing the cryptographic key in a second hardware register of the decryptor;

e. forming an initial counter value from the nonce value; and

f. sending the initial counter value to the decryptor, wherein the CPU resumes operations after the decryptor initializes.

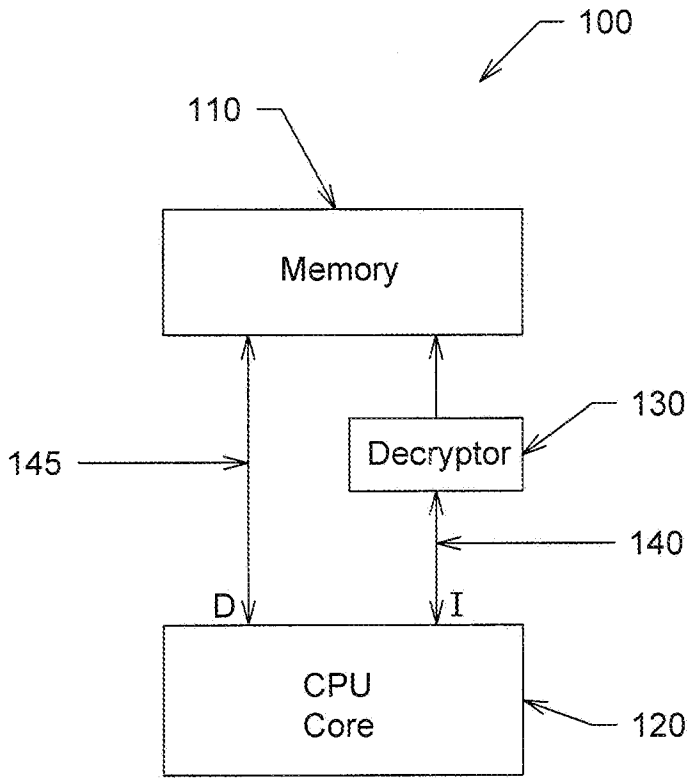


Figure 1

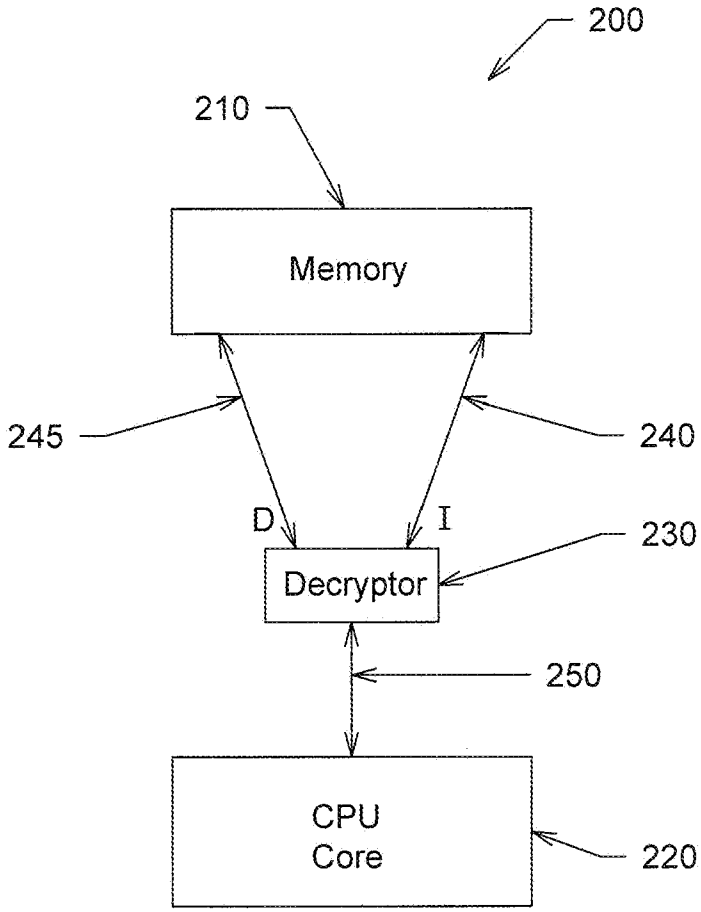


Figure 2

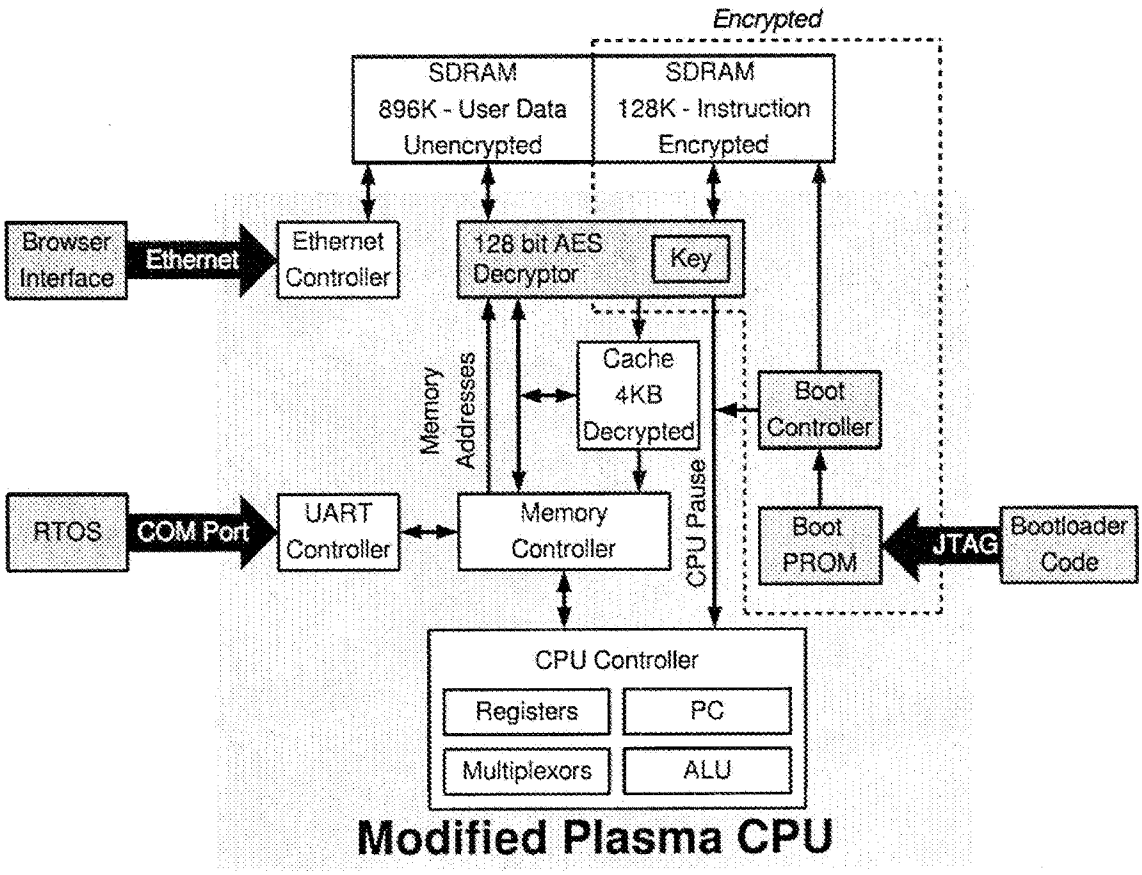

Encrypted
SDRAM
896K - User Data
Unencrypted
SDRAM
128K - Instruction
Encrypted
Browser Interface
Ethernet
Ethernet Controller
128 bit AES Decryptor
Key
Memory Addresses
Cache
4KB
Decrypted
Boot Controller
RTOS
COM Port
UART Controller
Memory Controller
CPU Pause
Boot PROM
JTAG
Bootloader Code
CPU Controller
Registers
PC
Multiplexors
ALU
Modified Plasma CPU

Figure 3

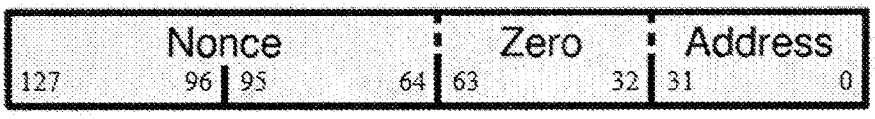


Figure 4

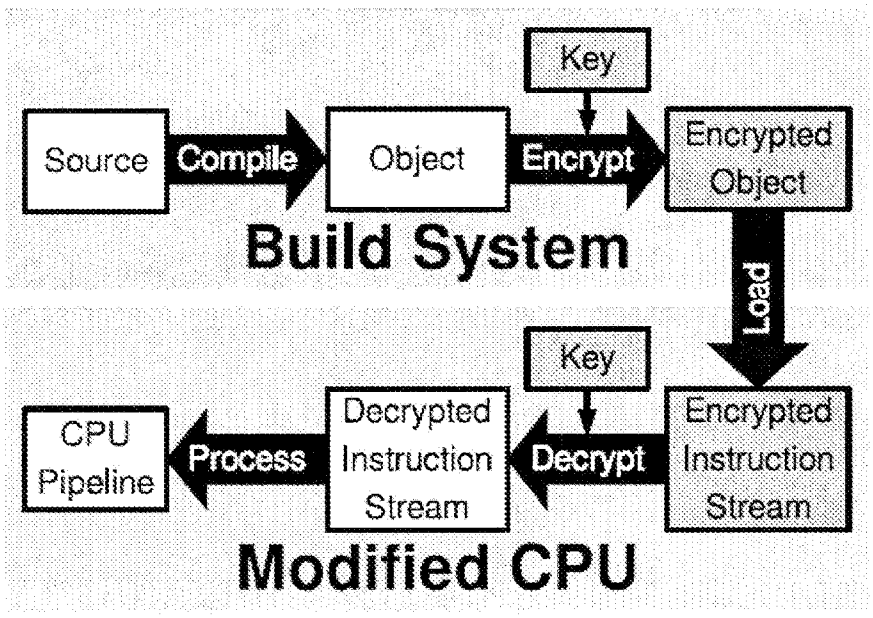


Figure 5

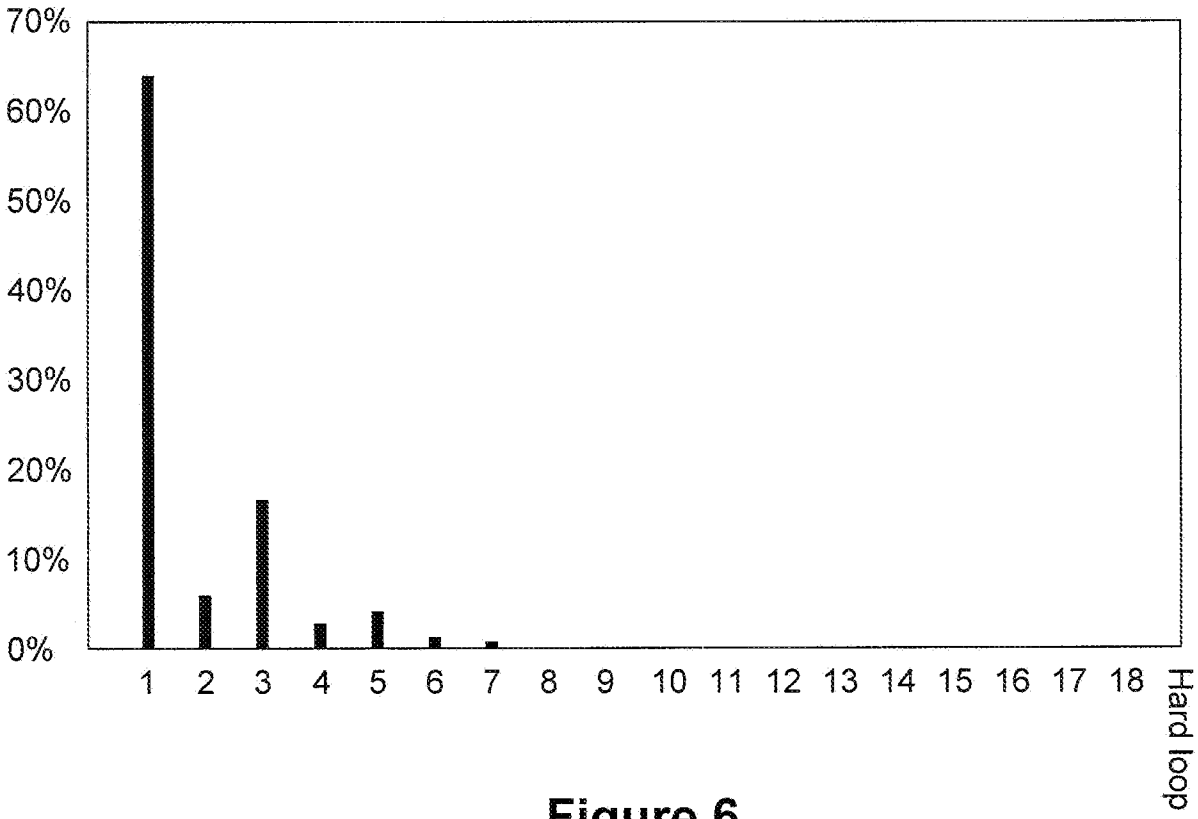

70%
60%
50%
40%
30%
20%
10%
0%
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
Hard loop

Figure 6

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2014/036675

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F21/72 G06F21/76
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2011/296202 A1 (HENRY G GLENN [US] ET AL) 1 December 2011 (2011-12-01)	1-4, 22-24, 41,44
Y	abstract; figures 1-2, 8, 27 paragraph [0004] - paragraph [0007] paragraph [0040] - paragraphs [0053], [0070] - [0072]	5-21, 25-40, 42,43
Y	US 4 465 901 A (BEST ROBERT M [US]) 14 August 1984 (1984-08-14) column 3 - column 9; figures 8-9	5,14-21, 25,34-40
Y	US 2010/246814 A1 (OLSON CHRISTOPHER H [US] ET AL) 30 September 2010 (2010-09-30) paragraph [0007] - paragraphs [0009], [0077], [0238]; figure 10	5-21, 25-40, 42,43

-/--

[X] Further documents are listed in the continuation of Box C.

[X] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
22 December 2014	09/01/2015

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Ghani, Hamza

1

Form PCT/ISA/210 (second sheet) (April 2005)

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	PETER T BREUER ET AL: "A Fully Homomorphic Crypto-Processor Design", 27 February 2013 (2013-02-27), ENGINEERING SECURE SOFTWARE AND SYSTEMS, SPRINGER BERLIN HEIDELBERG, BERLIN, HEIDELBERG, PAGE(S) 123 - 138, XP047032785, ISBN: 978-3-642-36562-1 the whole document	21,41
Y	WO 2008/054456 A2 (LUNA INNOVATIONS INC [US]; VIRGINIA TECH INTELL PROP [US]; GRAF JONATH) 8 May 2008 (2008-05-08) paragraph [0005] - paragraph [0015]	41,44
Y	KC G S ET AL: "Countering code-injection attacks with instruction-set randomization", PROCEEDINGS OF THE 10TH ACM CONFERENCE ON COMPUTER AND COMMUNICATIONS SECURITY; [ACM CONFERENCE ON COMPUTER AND COMMUNICATIONS SECURITY], WASHINGTON D.C., USA, vol. CONF. 10, 1 January 2003 (2003-01-01) , pages 272-280, XP002333430, DOI: 10.1145/948109.948146 ISBN: 978-1-58113-738-5 the whole document	21,44
A	Ana Nora Sovarel ET AL: "Where's the FEEB? The Effectiveness of Instruction Set Randomization", 14th USENIX Security symposium, 31 July 2005 (2005-07-31), XP055159765, Retrieved from the Internet: URL:https://www.usenix.org/legacy/events/sec05/tech/full_papers/sovarel/sovarel.pdf?CFID=464936044&CFTOKEN=31104110 [retrieved on 2014-12-19] the whole document	1

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/US2014/036675

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2011296202 A1	01-12-2011	CN 103645885 A	19-03-2014
		CN 103699832 A	02-04-2014
		CN 103699833 A	02-04-2014
		CN 103713883 A	09-04-2014
		CN 103761070 A	30-04-2014
		CN 103839001 A	04-06-2014
		TW 201419142 A	16-05-2014
		TW 201426537 A	01-07-2014
		TW 201426538 A	01-07-2014
		TW 201426539 A	01-07-2014
		TW 201426540 A	01-07-2014
		TW 201426541 A	01-07-2014
		US 2011296202 A1	01-12-2011
		US 2011296203 A1	01-12-2011
		US 2011296204 A1	01-12-2011
		US 2011296205 A1	01-12-2011
		US 2011296206 A1	01-12-2011
		US 2012096282 A1	19-04-2012
		US 2014195820 A1	10-07-2014
		US 2014195821 A1	10-07-2014
		US 2014195822 A1	10-07-2014
		US 2014195823 A1	10-07-2014
US 4465901 A	14-08-1984	NONE	
US 2010246814 A1	30-09-2010	NONE	
WO 2008054456 A2	08-05-2008	US 2010122095 A1	13-05-2010
		WO 2008054456 A2	08-05-2008