Xbox-Next Security

Only Fools Make Predictions

Sufficient rumors have been floating around on the Internet to start painting a picture of what the Xbox-Next security may look like. This paper is all speculation, and none of it is grounded in hard fact, so take it with a grain of salt.

A good way to start the speculation process is to first determine the possible parameters for the next-generation security scheme. I am assuming the parameters for Xbox-Next security is:

- Minimal silicon area/parts count
- Must not degrade production yield or otherwise increase manufacturing costs
- Must be resistant against hack-once, run everywhere attacks
- May be weak against one-time "hero" hacks
- May use custom features in the processor or chipset cores

Let's also recap what is known to date.

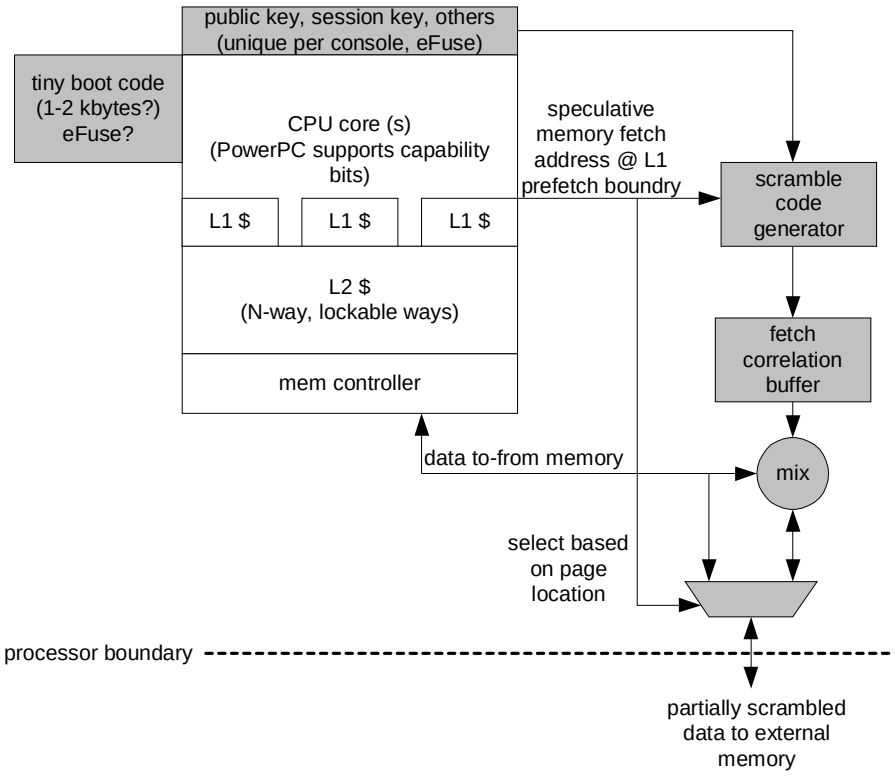
1) Xbox-Next will be built around the PowerPC architecture by IBM. In fact, it will be built around three PowerPCs, perhaps all on one chip, perhaps not.
2) Xbox-Next will use an ATI R500-type graphics device. It looks like it will likely be a custom job as well.
3) There will be a large amount of external memory (128-256 MB, depending on which rumors you trust)

Given these parameters, I believe Xbox-Next security will involve some combination of the following elements (not all may be implemented):

1) Unique public key and probably other unique data per box, stored inside the CPU silicon.
2) Boot/kernel code in FLASH ROM encrypted using a unique session key, also with hash signed by a unique private key (private key is not contained in Xbox-Next)
3) Secure bootloader power-on reset program contained within the processor silicon
4) Decryption and authentication of boot/kernel code occurs entirely within the processor's L2 cache using lockable ways
5) Partially encrypted/scrambled memory space, using page-selectable encryption/scrambling. This allows for code to be encrypted but other data, such as textures and sound data, to reside unencrypted and accessible to regular DMA driven devices

6) Encryption on memory space is probably closer to scrambling than encryption for performance reasons. However, the key used for scrambling is unique per console.
7) Use of pointer tag bits or execute-only code pages. Specially tagged address words and/or code pages make buffer overrun attacks extremely difficult, if not impossible. The Power architecture supports capability bits; and if the PowerPC architecture does not already support execute-only code pages, it is probably a relatively simple matter to extend the page table mechanism to include this feature.

The following figure illustrates the gross arrangement of these features within the Xbox-Next processor chip(s):



Some general observations about this organization:

- This requires no custom silicon other than the microprocessor
- Area impact within processor is minimal. Most crypto operations are done using the CPU. The only operation done entirely in hardware is the scramble code generator and the fetch correlation buffer. I explain how this might be done later
- I will also explain how the keying of this system will happen in a manner that is production and diagnostic-friendly

Now, I will talk about each of the security elements and how they might be done, and what they might mean.

Unique Crypto Keys Per Processor

This feature is the most important security feature for Xbox-Next. The ability to assign a unique key to each Xbox-Next unit is an important enabler to defeating almost all types of hack once, run-everywhere attacks. The ability to encode a public key of a PKA pair enables Microsoft to ship Xbox-Nexts without all the secrets necessary to author new code for the Xbox as well. Let me talk first about how this uniqueification would be implemented.

Recently, IBM announced the availability of an electromigration-based fuse technology called eFuse. This technology is field-programmable, dense, and extremely cheap—it relies on properties inherent in all chip back-end processes. I'm guessing it would cost minimal area to pack in about 2000 bits of eFuse information on a chip. For reader reference, an EE Times article quotes a bank of 2000 fuses being used to repair a 16 Mbit DRAM chip, and that eFuse bays come in blocks of 128, represented at the design phase by a VHDL block that just drops in on the chip. I am predicting that Xbox-Next CPUs will roll off the IBM (or whoever is fabbing the chip) line with about 2000-5000 bits of information that is eFuse programmable. Of course, the chip is shipped initially to Microsoft as a blank. During the blank condition, all of the cryptographic features are disabled and the processor operates as a normal processor would. This helps satisfy a key condition: low yield impact in system assembly. Because Microsoft can fully-assembly Xbox-Nexts and have them remain generic, defective Xbox-Nexts can be more easily diagnosed and repaired.

Once an Xbox-Next assembly is tested and validated, the FLASH ROM is burned with an initialization program. This init program first puts the CPU to work creating its own public/private key pair. The init program then uses the network port to download a raw boot/kernel image, and encrypts this boot/kernel image using a randomly generated session key. This image and its signature is then burned into the FLASH ROM. Yes, I believe that the FLASH ROM will be write-able in Xbox-Next, because with the proposed security in this paper, there is no point in hard disabling the write-ability of the FLASH ROM by the CPU—read on. The public key and session key are burned into the CPU's eFuse, and the private key is transmitted to a Microsoft database and never seen again, or at least not until Microsoft wishes to securely transmit a field BIOS upgrade via Xbox-Live using this public/private key pair. eFuses, by the way, take 200 microseconds each to program and 10 mA of current. At 200 microseconds, a bank of 2000 fuses can be programmed in around 0.4 seconds. In fact, at this size and at these speeds, it is quite possible that the initial secure bootloader for the Xbox-Next may be stored on the chip entirely using eFuse technology, thus allowing much more flexibility on Microsoft's part in terms of rotating the secure bootloader if an exploit is discovered.

The console is now ready for boxing and shipping to the user.

Once the Xbox-Next has been appropriately keyed, a small internal bootloader within the CPU core becomes active. This bootloader does something of the following (some steps are optional):

1) Use the unique-per-console session key to decrypt unique-per-console-encrypted boot image
2) Use public key to verify hash signature of boot image
3) Assuming all checks out, jump to decrypted boot code

Let's recap what this procedure has accomplished.

- Each Xbox-Next has a unique session key and every ROM image is signed using a unique digital signature, thus defeating crack-once run-everywhere to some extent
- Microsoft retains control over the private key, and never discloses it to anyone.
- This means that even if you were able to read every transistor off of the CPU die, you still could not encrypt your own valid FLASH ROM (hence there is no need to write-protect the FLASH ROM), unless you can break the PK or hash algorithm used by the Xbox-Next
- Xbox-Nexts roll off the production line completely generic ("blank") and fully diagnose-able, thereby keeping assembly and rework costs low; customization and uniqueification only happens after the Xbox-Next is fully tested and verified.

Plugging the Memory Hole

Sounds like pretty tight security, huh? Well, there are a few problems still with the procedure outlined above. The most important one is that the decrypted BIOS code eventually gets written out to main memory. This is a key weakness for a hack-once, run-everywhere hack.

For example, suppose a "hero effort" involved snooping all traffic to main memory. This hero effort would yield two things:

1) kernel code, which can be analyzed for exploits
2) a set of timings with regards to when certain pieces of code are written to or read from memory

Item (1) is bad, but there are ways to mitigate that problem-discussed later. Item (2), however, is potentially deadly. If the attacker could determine the precise number of clocks since reset (or perhaps a very unique bit pattern that can be easily observed) in which to flip a single bit in an instruction stream going into or out of main memory, the attacker could corrupt a jump destination to point into malicious code. Such an attack could be implemented using a simple CPLD and perhaps as little as three wires.

There are two things Microsoft needs to do to prevent this attack.

First, Microsoft must implement its initial decryption and hash verification of the boot code using only L2 cache memory. It is quite likely that a significant amount of code could be locked into L2 cache for this purpose (looking at the Power4 dual-core architecture, I'd say that a shared L2 cache would have to be pretty large anyways). Furthermore, since the Nintendo Gamecube's PowerPC supported lockable L2 cache ways, it is likely that the Xbox-Next's CPUs will also support this feature, useful for squeezing extra performance out of games. Once the L2 cache is locked, the entire boot validation process can occur safely inside the processor, outside of the reach of contaminated busses.

Second, Microsoft could introduce selective scrambling on code pages. This would work by first computing a hash of the public key, or otherwise obtaining some console-unique pseudo-random number. Then, whenever a miss happens in L1 cache, the address of the miss is observed. If it is within a code page, then a key is prepared using a combination of the console-unique pseudo-random number, the exact fetch address, and a little bit of computation. By the time the request propagates down to the main memory interface, the key is fully ready, and it is mixed with the outgoing or incoming data request/reply. Thus, the effective data access latency penalty of this operation is the mix operation, since all of the keying and coordination operations can happen in parallel with the cache lookups. The mix operation could be as simple as an XOR, although that is possibly a bit too weak. An operation that involves some bit diffusion (so that a single bit plaintext change affects multiple ciphertext bits) would be desirable for reasons described later.

The reason only some pages can be encrypted is that it is undesireable to require all DMA devices to also employ the same scrambling/encryption scheme. Also, there is less reason and probably less benefit to encrypting non-code segments, especially when combined with techniques listed in the next chapter that guard against buffer overrun attacks and rogue pointer creation. Also, most DMA data is not critical game data—they are textures, rendering commands, audio samples, etc. It is probably okay to push those around in the clear—or rather, even if it is a weakness to manipulate such data in the clear, performance requirements would have it no other way.

However, as seen in the Xbox, memory layouts in the Xbox console are relatively static, so it is quite feasible to designate a region of memory as "just code" and everything that goes there should be encrypted.

Significantly, the "encryption" used on this data does not need to be that strong (or so I think…), which is why I refer to it as a "scrambling" rather than as an encryption. The point is not to hide the data well. Rather, the point is to defeat or at least complicate a break-once, run-everywhere attack. Once the code is scrambled going into main memory, it is more difficult to develop a simple modchip that can patch memory transactions.

Plugging the Exploit Hole

A little known feature of the Power architecture is its "tag bit", or 65th bit (see http://lwn.net/2001/features/OLS/pdf/pdf/iseries.pdf). This bit has the property such that setting it designates the word as a pointer, and this bit can only be set by the kernel. Therefore, it is impossible for user-space code to create, intentionally or accidentally, an arbitrary pointer into memory. If Microsoft opts to use this bit in the Xbox-Next kernel, it can increase its resistance to many kinds of exploits quite dramatically and perhaps outright eliminate certain classes of software attacks.

Another feature that Microsoft could have implemented, if it is not already available in the Power architecture, is a page-based code-only bit. If a page of memory has this bit set, then it contains code, and is therefore read-only. This prevents programmers from accidentally bashing code space, and it also prevents hackers from modifying and patching code. So long as only the kernel has the ability to create code pages, user-space exploits will have a difficult time turning data loaded in via a buffer overrun into executable code.

Note that the decrypt-and-validate in cache and the scrambled code in memory concepts are extremely important in enforcing all of the above software security ideas, because it assumes integrity of the kernel. If memory space can be arbitrarily tweaked by hackers through a man-in-the middle attack on main memory, then a weakness can be introduced anywhere in the system—and the whole thing falls apart.

FAQ

1) No, I am not planning on cracking the Xbox-Next. I wrote this paper because I couldn't fall asleep the other night and I had a few ideas sloshing around in my head that I needed to get down on paper.
2) This paper is purely speculative. No, I have not seen the Xbox-Next security specs. Perhaps there is no security on the Xbox-Next. Perhaps the security is way stronger than what I am predicting. Perhaps it uses a completely different system. We'll find out in about two years.
3) I am guessing that someone out there made a lot of money (millions of dollars) selling Xbox modchips. That guy would have the finances required to completely map out the Xbox-Next silicon and develop a good crack. I figure it shouldn't be too long before an effective modchip is available on the market.
4) If you have any thoughts or insights/inspirations, please feel free to share them with me. I find idle speculation to be entertaining.