Hardware Implementation of Secure Communication in a Bus-Based Multi-Core System Using Tiny Encryption Algorithm

Team DJA DJA

Proposed Solution to Unsecure inter-core communication:

We propose to solve the problem of un-secure inter-core communication by using a hardware encryption method to encrypt data which is transmitted via the bus. Each time a core requests to read contents of the memory from other any other core, the data will be encrypted and then transmitted on the bus. TEA (Tiny Encryption Algorithm) is a simple and fast symmetric encryption method. In the process of encryption and decryption, only addition, subtraction and XOR are involved, which make it a good hardware encryption method. In all, 16 rounds of encryptions will be executed (we can run multiple rounds in one clock cycle if core clock is slow enough compared to the computation latency). Since TEA is quite simple to implement, it may not take very long to use brutal force to hack into the bus and snoop on the data. So the target here is to renew the keys periodically to avoid hacking using brute force. We will use a centralized key generator & distributor to implement this. The generator will use random numbers to generate random new keys. We can estimate the number of cycles required using the brute force method to hack into the system, using this data, set the refresh period of the key-generator smaller than this time period.

Progress:

Owing to the difficulties involved with implementing the Wishbone bus using the version of VCS used in CAEN computers, instead of using or1200 core, we switched over to a simple multi-core system (previously designed for EECS 470). This system provided us with 2 cores, connected together by a simple bus, and having 2 levels of cache implementing coherence protocol.

However, since the bus of the 2-core system is not scalable (due to the unique cache coherence protocol and some other control signal between the caches and bus), we have changed our plan from a 4 core 1 memory system to a 2 core 1 memory system, together with a dedicated simple hijacking core beside (this core will only try to decrypt the encrypted data which is being transmitted on bus, and will not communicate with other cores). Another major modification to our scheme is that the encryption/decryption time has been shortened. The clock period of the system is 10ns, and the encryption/decryption module only requires 1.25ns to finish one round of computation. In other words, we can put 8 rounds of computation within one clock cycle, and since in total TEA requires at-least 16 rounds of computation, the entire encryption/decryption process can be finished within 2 clock cycles.

Analysis:

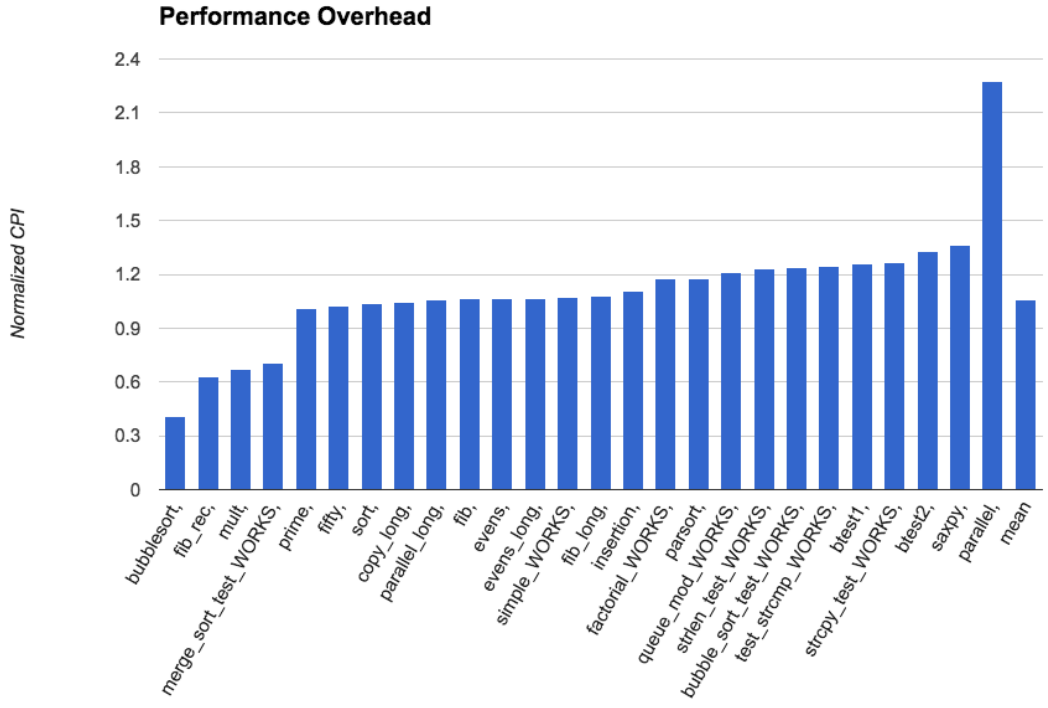
The brute force method was tried out to hack into the multi-core system. As we are changing our key at a frequent rate, brute force actually does not make any sense, and is carried out just for analytical purposes, and finding the time period of refreshing the keys. For this analysis, the keys are to be kept constant. Since the key is 32 bits long, it takes maximum 4 billion cycles for all the combination of keys to be tried out. Assuming each encryption/decryption process takes 2 cycles, and considering that all the encryptions/ decryptions are happening in parallel, it takes maximum 8 billion computation cycles for the brute-force procedure to complete.

Assuming that the keys are being refreshed after a very relaxed interval of 1000 cycles, the probability that brute force will be able to attack the system decreases even further. In numbers, the probability that brute force can hack into our system is of the order of $10-7$ Hence, we can conclude that our proposed technique is resilient to brute force attack.

Result:

After integrating our modules to the two-core system, we measured the performance overhead against the original design. The same tests were run on both the new design and the original design, and the CPIs of our design were normalized to the ones of the original design. Very diverse and interesting results were obtained. A maximum overhead of 2.27 (parallel) and a minimum of 0.41 (bubblesort), with a geometric mean of 1.06 was observed for our design.

It is quite intuitive that the performance would degrade since delay elements are added for memory access. Advanced features, such as prefetcher, write-back cache, out-of-order execution etc, can reduce the impact of this added latency. For some tests, improvement in performance were observed (bubblesort, fib_rec, mult and merge_sort_test_WORKS). This might be since the memory access patterns are changed and caches in these tests expose better locality, which can reduce miss rate and thus improve performance.



Next Steps:

We are working on developing another attack module, more intelligent than brute force, which can hack into our module at a rate faster than brute force, so that we can have a more accurate indication for selecting the value of the number of cycles after which the key should be refreshed.