Siddhant Shah
COMS 4187

1.

The columns represent the AES inputs, and the key that was applied to them represents the columns. The colors show how long the encryption of that 4x3 block took on average, in terms of cycles. The goal is to make it so that the amount of time the encryption took is not dependent on the key. Thus, to be secure against timing attacks, the graph should show vertical stripes. That would show that the AES encryption time was not dependent on the key.

Going by that, the more disordered and irregular, then the more vulnerable to a timing attack, since correlations can be made between different times and different keys. OpenSSL seems to perform poorly all around, as seen by it's chaotic hodge-podge. Gladman does better, showing more regularity. AMD and Pentium III look worse by the same paradigm than Pentium M, IBM PowerPc, and Sun Ultrasparc.

2.

A Trusted Platform Module is a small hardware device that comes with crypto capabilities of its own. One of its potential functions is deciphering encrypted communications sent to the platform it is housed in, without ever disclosing its key to the platform. Another is verifying the that the platform itself is legitimate to the applications the TPM interacts with. It can do this by monitoring characteristics of the platform's usage. Aside from whatever API exists between the platform and the TPM, the platform has no reach into the inner workings of the TPM.

A TPM is essentially like a third party that lives on the platform, hiding information from the end-user and even barring usage if configuration parameters change. This can be useful in cases where the platform is infected with malicious code that tries access encrypted data. However, TPMs present a large risk where the manufacturer has to be trusted not to collude with parties that would want the TPM keys to decrypt your data.

3.

Federated authentication is a multiple sign-in to multiple site architecture. However, this method can cause a lot of user frustration, as it requires multiple account setups, reiterative input of data, and the headache of remembering increasingly long passwords. However, the data breach shows that the pain is necessary for security. Now, with 50 million accounts compromised that have logins to more than one service, the damage for a single user can be even greater than the sum of the parts. Due to all the data that a hacker could gain, it may be feasible for them to gain access to even more accounts that were not under Facebook's umbrella.

The breach reinvigorates the desire for federated authentication. Depending on the inconvenience imposed on the other services, they may drop Facebook's single sign on, or at least halt new users from joining that way. Users themselves may elect to create new accounts and do the overhead if it keeps them secure. Apparently damage has to be done before security can really be appreciated.

Programming
Results:

Function	OpenSSL (C++) (ms)	PyCryptoDome (Python) (ms)
AES128 Encrypt	0.0008223	0.0004298
AES128 Decrypt	0.0004121	0.0003819
SHA256 1e2 Bytes	0.0013779	0.0028090
SHA256 1e3 Bytes	0.0073838	0.0050528
SHA256 1e4 Bytes	0.0700748	0.0531001
SHA256 1e5 Bytes	0.7017968	0.5235541
SHA256 1e6 Bytes	7.0244408	3.0592101
RSA 1024b Key Encrypt	0.0228231	0.0695281
RSA 1024b Key Decrypt	0.1585003	2.1834361
RSA 2048b Key Encrypt	0.0410398	0.1889059
RSA 2048b Key Decrypt	1.0078863	10.602839

Random Number Generator	Python Normal	Python Secure
16 Bytes	0.000017095	0.001337601

Analysis:

10,000 trials were done with each function, and the average time to execute the aforementioned operations is shown in the table. For the python results, the best of a set of 10 separate 1000 long runs was used, because python generally overestimates the time it uses.

On average AES128 encrypt took 0.0008223 ms, and 0.0004121 to decrypt via OpenSSL. For PyCryptoDome, it took 0.0004298 ms to encrypt, and 0.0003819 ms to decrypt. Interestingly, AES decryption appears to be quicker than AES encryption, which appears to be counter to Rjindael's design. It is likely due to the implementations. For AES, PyCryptoDome was faster.

For SHA256, OpenSSL appears to be faster for lower numbers of bytes. However, larger sequences generally matter more, so PyCryptoDome wins out in this case. Hashing time seems to linearly increase with input length, as expected since only 256 bits can be hashed at a given time with SHA256.

For RSA, OpenSSL was much faster than PyCryptoDome. Decryption took much longer than encryption, about 2 orders of magnitude longer in either case. The private key files were much larger than the public key files.

As for the random number generator, using the standard random python library was 2 orders of magnitude faster than using the new Python secrets module, which generates crytographically secure random numbers.