Mark Weston 5/12/05

Professor Almstead: CSc 197

Background Paper: The Current State of Cryptography Implementations

Cryptography's main strength lies in its solid mathematical foundation, a foundation that is independent of implementation. However, with the advent of the computing age, cryptography has been adapted and implemented in computers to secure transmissions and protect data from prying eyes. As my project proposes using the RSA and El-Gamal cryptosystems in the context of an online tool, it would be useful to explore the state of current implementations. Implementation difficulties will be explored, as well as the feasibility of attacking these two systems. This research will enable a project that will provide greater levels of functionality to the user.

As mentioned above, the mathematical basis of cryptography is solid. The currently successful and popular cryptosystems have been subjected to a rigorous peer review process. Many of them are arguably unbreakable within a 'reasonable' amount of time. By 'reasonable' I mean a finite amount of time an attacker would be willing to invest, given current hardware. Unless we see an epic increase in processing power or a significant advancement in number theory, the length of time to complete many of these attacks will continue to be measured in years [4].

If this is true, we should be living in a privacy dream world. Why then, do some mistrust the protection that cryptography offers? Many are very wary of giving out personal information online, even to recognized parties. Online shopping is convenient, but who knows what is *really* happening to your credit card number as it sails out across the Internet. The problem has to do with the current implementations of recognized cryptosystems [2].

The mathematics behind cryptography gains considerable strength by using problems that, by principle, are difficult to solve. The implementations of those problems, however, lack the same kind of validation and proof. They are done by people, people who can and do make mistakes. In addition to the cryptosystem itself, there is a massively complex 'context' that is needed to provide services to end users [2]. The lesson here is that cryptographic algorithms and protocols, if implemented and used correctly, can be a powerful aid to a software system. Although I will be focusing on a single user, it is important to realize is that implementation can be tricky.

One example of an implementation difficulty could be prime generation. Although normally the generation and testing of very large prime numbers can can be difficult, some lenience is given here due to the probable requirements of the project. As the problems assigned in the course are meant to be worked out by students, by hand, the primes needed are unlikely to exceed 4 decimal digits. However, if more advanced functionality is needed, it may be worthwhile to allow for the creation and testing of large prime numbers. This could be accomplished by starting with a large number, then iterating it and successively performing primality tests on it to see if it is 'probably prime'. This will usually result in numbers that are very unlikely to be composite, usually not exceeding a chance of 1 / 2 ^ 100. There are a variety of algorithms available to do this, with varying speeds and difficulty of implementation [5]. This subproblem of prime generation and prime testing proved to be very interesting and could be used as a way to add 'Computer Science' rigor to the project.

A second area of difficulty for implementation lies in the modular arithmetic central to one of the cryptosystems to be used – RSA. Specifically, a fair amount of modular division and exponentiation need to be performed when performing this method. Java provides a useful math class called BigInteger that provides these methods. In addition to methods dealing with modular arithmetic, other useful cryptographic methods are implemented, such as GCD and prime generation [6]. Although it probably won't be needed, this class will be available if I'm looking for a way to test my implementations. This will probably not be used in my project simply because its great strength – a powerful class library, is offset by its inflexibility. If something needs to be changed or done in a different way, I would be stuck with what the class gives me.

A third area of difficulty for implementation has to do with picking/generating some of the values used in the RSA cryptosystem – *p, q, n, e,* and *d*. Although it has been mentioned before that the requirements of this tool will be less strenuous than most, flexibility should be required. A truly successful implementation should be able to generate and work with large primes *p* and *q*, a large modulus *n*, and encryption and decryption constants *e* and *d*. Poorly chosen values may still accurately decrypt/encrypt, but could be indicative of an underlying weakness or fault. Poorly chosen values of *n,* for example, could lead to it being factored by several well known methods. Because RSA's strength relies on the belief that no efficient factoring algorithms exist for suitably chosen values of *n*, the implementation will be inherently weak [1]. It is important to note then that in principle, a weak cryptosystem can be just as bad from an implementation standpoint as a flawed or non-working one.

Why is this important to consider? As we are starting to see, cryptosystems that fail often do so not because of cryptanalysis, but because of implementation and management errors. This should be considered when deciding upon the scope of my project. A faulty cryptosystem has little or no value to anyone. It will be beneficial to remain open about the project so that the users will be able to given feedback, as well as receive information from the author about what the system can and cannot do.

However, assuming these implementation issues can be tackled, the resulting system should be very effective. By that I mean it will behave like a normal cryptosystem would. If large values are used, the math behind the systems will create effectively one-way functions. Text will be able to be easily and quickly encrypted and decrypted, but without extra knowledge about the problem, standard attacks will not be successful [1], [4]. However, if small values are used, these same attacks could be implemented as a proof-of-concept of how they work on vulnerable cryptosystems. A useful metaphor for explaining this is that of an attacker who would like to enter a certain territory, however, there is a wall blocking him/her. The attacker can be envisioned as one of many algorithms targeting the mathematical basis of the cryptosystem, hoping to eventually solve it. The wall is the cryptosystem. If small values are used, the wall is small. Then, the attacker, who can only climb at a very slow rate, will be able to reach the top. However, if large values are used, the wall gets exponentially higher, and nothing the attacker can do will increase the rate of his/her ascent. It will be important to illustrate this concept and include the flexibility for both relatively large and small values when using the tool.

I mentioned earlier that the mathematical foundation of these two systems is solid. I would like to take a moment now and expand on that just a little. The first system used, RSA, only exposes an an encryption key, *e*, and a modulus, *n*, to the public. An attacker will be able to successfully 'break' RSA if they can find the private decryption key, *d*, which is related to the above two terms by the equation:

$$ed \equiv 1 \, mod \; phi(n)$$

where n is equal to the product of two large primes, *p* and *q*, and phi(n) = (p-1)*(q-1). Given phi(n), an attacker can solve the above relation easily. However, phi(n) can only be obtained by factoring n, obtaining p and q, and then computing phi(n). It has been proposed, then, solving the above relation is equivalent to factoring n [1]. Fortunately, this has been shown to be a difficult problem to solve, given large values of p and q. RSA Security is currently holding a contest to factor “n's” with varying bit and decimal lengths. Cash prizes are awarded, the highest of which is $200,000 for a 2048 bit modulus. The largest current value that has been successfully factored uses 576 bits. It was posted in July of 2001 and was broken in December of 2003. This was accomplished using distributed research centers across several countries over a period of two years [3]. Using today's hardware and algorithms, if it were feasible, the higher moduli would have been cracked by now in order to win those cash prizes. However, the current state of factoring has shown that even though RSA can scale well into 200 digit decimal numbers, it is not feasible to factor those numbers at this time. It takes an epic amount of time and resources to break a single communication that may or may not be of worth.

The second system, El-Gamal, uses a different defense, but is similarly strong. The El-Gamal system uses a problem known as the discrete-logarithm problem. It also uses a large prime, *p*, a "generator" *g* of p and a value *a*, that is an element of the set g can generate. g has the property that when repeated exponentiation is performed, followed by modulo division of p, it yields a set of unique elements <= p-1. The goal then, is to find a value *i*, such that

$$a \equiv g^i \, mod \, p$$

The normal brute force method is to apply all j from 0 to p-1 to *g* to see if it matches *a* after the modulo division. However, for large primes, this is infeasible. Additionally, prior work does not improve the possibility of future calculations being successful. The exponentiation and the modulo division disguise each result from the next, until the correct 'i' is chanced upon [1]. Doing a few simple examples by hand can assure the user of this.

At this point we have explored the state and feasibility of current implementations of the two cryptosystems to be used. Some implementation difficulties have been explored, such as prime number generation, modular arithmetic, and picking appropriate values for the systems. The safety of these two systems has also been examined. If appropriate values are picked and the implementations are correct, both systems provide exceptional defense against cryptanalysis attacks. Being aware of these issues and the properties of the systems I intend to work with should help produce a better designed, easier to use, and more functional project. If I can complete the 'core' resources in a quality manner in less time, I will be able to spend more time adding other features, as well as working on the user interface.

References:

[1] K.M. Pradosh, “Public key cryptography,” *Crossroads*, vol. 7, no. 1, 2000, pp. 14-22.

[2] P. DePalma et al, “Cryptography and Computer Security for Undergraduates,” 35th Technical Symposium on Computer Science Education (SIGCSE), pp. 94-95.

[3] “RSA Security – The New RSA Factoring Challenge” May 2005; http:// www.rsasecurity.com/rsalabs/node.asp?id=2092

[4] R. D. Silverman, “Massively distributed computing and factoring large integers,” *Communications of the ACM* vol. 34, no. 11, Nov. 1991, pp. 95-103.

[5] E. Shade, “Ready for prime time?,” *Journal of Computing Sciences in Colleges* vol. 17, no. 3, Feb. 2002, pp. 282-289.

[6] "Sun Java API Documentation" May 2005; http:// java.sun.com/j2se/1.5.0/docs/api/index.html