An Efficient Parallel Algorithm for Secured Data Communications Using RSA Public Key Cryptography Method

Christofer Chávez Carazas

6 de junio de 2017

Motivación

- Utilizar *GNU MP* para realizar las operaciones con enteros grandes que requiere el *RSA*.
- Paralelizar la exponenciación modular con *OpenMP* para reducir el tiempo de encriptado y desencriptado. del *RSA*

RSA

- Sistema criptográfico asimétrico.
- Producto de dos números primos grandes.
- Usa al exponenciación modular.

Generación de llaves

1. Se escogen aleatoriamente dos números primos grandes q y p.
2. Se obtiene el módulo $n = q * p$.
3. Se calcula $\varphi(n) = (p-1)(q-1)$
4. Se selecciona un e que cumpla: $1 < e < \varphi(n)$ y que sea coprimo con $\varphi(n)$.
5. Se selecciona un d que cumpla: $1 < d < \varphi(n)$ tal que $e * d = 1 \; mod \; \varphi(n)$.

Cifrado y Descifrado

Cifrado

$$c \equiv m^e (mod\ n)$$

Donde c es el mensaje cifrado, m es el mensaje original, e es la llave pública y n es el módulo hallado anteriormente.

Descifrado

$$m \equiv c^d (mod\ n)$$

Donde m es el mensaje original, c es el mensaje cifrado, d es la llave privada y n es el módulo hallado anteriormente.

Repeated square-and-multiply Algorithm

El algoritmo se basa en la siguiente observación:

$$g^e mod\ m = (g^{e/2} * g^{e/2}) mod\ m$$

y se puede definir recursivamente de la siguiente manera:

ModExp(g,e,m) =	**1,**	**if e = 0**
	(g X ModExp(g, (e-1), m)) mod m,	**if e is odd**
	ModExp(g, e/2, m)^2 mod m,	**if e is even**

Figura: *Repeated square-and-multiply* [S. Saxena et al. 2014]

Exponenciación Modular binaria Paralela

- Se convierte el exponente a su forma binaria y se lo divide en k partes dependiendo del número de procesadores.
- La exponenciación tendría la siguiente forma:

$$g^e = g^{2^{r_k} e_k} \cdots g^{2^{r_2} e_2} * g^{2^{r_1} e_1}$$

siendo las r calculadas de la siguiente forma:

$$r_1 = 0, r_2 = \frac{n}{k}, r_3 = \frac{2n}{k}, \cdots, r_k = \frac{(k-1)n}{k}$$

- Cada parte se calcula con el *Repeated square-and-multiply Algorithm*

Experimentos - Test Case Sets

Test Case Set 1

- Diferentes tamaños de llaves: 128, 256, 512, 768, 1024 y 2048 bits.
- Mismo tamaño de mensaje: 5000 caracteres.

Test Case Set 2

- Mismo tamaño de llaves: 1024 bits.
- Diferentes tamaños de mensaje: 1000, 2000, 3000, 4000, 5000 y 6000 caracteres.

Experimentos

- Se tomó el tiempo de cifrado, descifrado y las suma de los dos anteriores más el tiempo que demora la generación de llaves
- Para cada caso se corrió el programa 25 veces y se sacó el tiempo promedio.
- Se calculó el *speedup* para los casos con 8 threads.

Experimentos - Resultados - Set 1

Key Size	Segment Type	1 core	2 cores	4 cores	8 cores	Speedup
128	Encryption Time	0.01744412	0.00771176	0.00495088	0.00477248	3.66x
	Decryption Time	0.01446656	0.00718892	0.0047454	0.00479988	3.01x
	Overall Time	0.03349776	0.01610124	0.01075044	0.01062064	3.15x
256	Encryption Time	0.05685824	0.02920536	0.01735212	0.0162128	3.51x
	Decryption Time	0.0531176	0.02808028	0.0175166	0.0163878	3.24x
	Overall Time	0.11153568	0.05862836	0.036021	0.0337508	3.31x
512	Encryption Time	0.32565408	0.09876696	0.096369	0.06402796	5.09x
	Decryption Time	0.32293712	0.11303592	0.09683764	0.08386048	3.85x
	Overall Time	0.65054268	0.2137714	0.19513912	0.1497552	4.34x
768	Encryption Time	1.0516946	0.53718348	0.2941692	0.26068648	4.03x
	Decryption Time	1.05282916	0.54098404	0.3156102	0.26486876	3.98x
	Overall Time	2.10927176	1.08250296	0.61428724	0.52960552	3.98x
1024	Encryption Time	2.3437544	1.2055034	0.65111192	0.56111836	4.18x
	Decryption Time	2.33940148	1.20698612	0.66276608	0.56557168	4.14x
	Overall Time	4.69310056	2.42224044	1.32380848	1.13549232	4.13x
2048	Encryption Time	16.83832248	8.66942184	5.45345	3.79430744	4.44x
	Decryption Time	16.83518192	7.98278924	5.39046672	3.79369864	4.44x
	Overall Time	33.75374976	16.74329772	10.916207	7.66389472	4.40x

Tabla: Resultados con el *Set 1*

Experimentos - Resultados - Set 2

Message Size	Segment Type	1 core	2 cores	4 cores	8 cores	Speedup
1000	Encryption Time	0.46757464	0.24420352	0.1278022	0.1122454	4.17x
	Decryption Time	0.46962944	0.2443334	0.13012468	0.11307884	4.15x
	Overall Time	0.94608372	0.49687084	0.26618736	0.23471736	4.03x
2000	Encryption Time	0.93499464	0.47531964	0.25300544	0.22485532	4.16x
	Decryption Time	0.9367404	0.47633848	0.25627944	0.22439264	4.18x
	Overall Time	1.88104592	0.96081168	0.51940344	0.45735516	4.11x
3000	Encryption Time	1.41060228	0.7255222	0.38410404	0.3369	4.19x
	Decryption Time	1.4089258	0.72689044	0.38451772	0.33783732	4.17x
	Overall Time	2.83086976	1.46097024	0.77717956	0.68406344	4.14x
4000	Encryption Time	1.87498956	0.98255104	0.52236468	0.44708772	4.19x
	Decryption Time	1.87756324	0.9866046	0.52224776	0.45145868	4.16x
	Overall Time	3.76213744	1.97925	1.05276452	0.9076462	4.15x
5000	Encryption Time	2.34797976	1.21547488	0.63996784	0.55835468	4.21x
	Decryption Time	2.35117576	1.22052832	0.6975152	0.559929	4.20x
	Overall Time	4.70871276	2.4440092	1.34563036	1.12672756	4.18x
6000	Encryption Time	2.8022342	1.47444388	0.7817876	0.66855896	4.19x
	Decryption Time	2.8096998	1.47175488	0.77560952	0.67615184	4.16x
	Overall Time	5.6216748	2.95597272	1.56562592	1.35219584	4.16x

Tabla: Resultados con el *Set 2*

Sapna Saxena, Bhanu Kapoor **An Efficient Parallel Algorithm for Secured Data Communications Using RSA Public Key Cryptography Method**, 2014.