# Optimización en la generación de claves para firmas en anillo, espontáneas y enlazables

José Luis Salazar
Dpto. de Ingeniería Electrónica y Comunicaciones
Universidad de Zaragoza
Email: jsalazar@unizar.es

José Luis Tornos
Dpto. de Ingeniería Electrónica y Comunicaciones
Universidad de Zaragoza
Email: jltornos@unizar.es

***Resumen*—El gran desarrollo de las Tecnologías de la Información (TIC's) y su gran aceptación por parte de la ciudadanía ha permitido que múltiples sistemas de votación electrónica se desarrollen en los últimos años. Uno de los grandes retos es conseguir que estos sistemas puedan emplearse con una gran variedad de dispositivos, por lo que los protocolos implementados deberán poder ser empleados tanto en dispositivos con gran capacidad de cálculo y memoria (portátiles y ordenadores de sobre mesa principalmente) como por dispositivos con unas características más limitadas (tablets y smartphones). En este artículo se muestran dos estrategias distintas para el cálculo, desde el propio terminal móvil de votación, de las claves con las que los usuarios participarían en un proceso de votación electrónica y las características ofrecidas por cada una de ellas.**

***Palabras clave*—Generación de claves, Firmas en anillo, Votación electrónica (*Key generation, Ring Signatures, eVoting*).**

## I. Introducción

El desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC), ha permitido implementar servicios en otros tiempos impensables como pueden ser la televisión a la carta o la telefonía móvil. Uno de estos servicios es la votación electrónica. Aunque podemos decir que todavía no existe una generalización del servicio (tampoco de la democracia), sí ha sido probado con éxito en varios países como son, entre otros, Suiza y Estonia [1], [2].

Por otra parte, con las nuevas arquitecturas de los terminales móviles la comunicación de datos se hace de una manera más liviana sin que la pila de protocolos de comunicación en Internet las constriña asfixiantemente. En el afán de simplificar todo lo posible el uso de la necesaria criptografía en las votaciones electrónicas, en este artículo proponemos un algoritmo de generación de claves para un sistema de votación portable en el que en una situación ideal, el votante pueda votar desde cualquier terminal móvil (propio o ajeno), sin una gran formación en TIC y con todas las garantías de seguridad telemática.

La restricción no es poca. Hay que tener en cuenta que de esta manera podemos dar acceso al votante a algún tipo de servicio, pero debemos dotar al terminal al que está conectado de todos las medidas de seguridad que requieren las comunicaciones implicadas en una votación electrónica y para ello dotarle (aunque sea de forma temporal) de capacidad de generación de las claves criptográficas necesarias. Para ello optamos por el modelo de una única entidad de confianza [3] que sea capaz de mantener el anonimato a través de una identificación previa.

Este modelo de una entidad de confianza basa el anonimato en el uso de firmas en anillo [4]. Además si queremos que el servicio sea portable, exigiremos que sean espontáneas; y para ser eficientes deberían tener una longitud fija (lo más corta posible). Luego una elección adecuada de firma para este tipo de votación electrónica es la propuesta por Wei [5]. Sin embargo, el aprovisionamiento de claves en este tipo de firmas no es trivial ya que las exigencias criptográficas hacen que el cálculo se haga desde el propio terminal móvil, obligándole a una carga computacional de la que no se está seguro se vaya a poder abastecer a priori.

En este artículo proponemos un modelo de generación de claves para este tipo de protocolo criptográfico que sea ejecutable en terminales con restricciones computacionales o de memoria, que pueda rellenar los campos de un Certificate Signing Request (CSR) de manera autónoma y almacenar la clave privada generada para su posterior uso en la votación electrónica. En el siguiente punto haremos una breve descripción del algoritmo de votación desarrollado a partir de las firmas en anillo y los requisitos que exigen sus claves. A continuación describiremos en profundidad nuestra propuesta para la generación de dichas claves, para finalmente exponer nuestros resultados y conclusiones.

## II. Escenario

El escenario que planteamos [3] contempla tres actores diferenciados:

- El votante, que en principio no cuenta más que con un terminal para comunicarse y hacer las operaciones pertinentes.
- Una Autoridad de Certificacion (AC) que comprueba la identidad del votante y le emite un certificado asociado a un CSR enviado por éste. Por otra parte sirve de repositorio confiable de los parámetros asociados a un evento de votación, imprescindibles para construirse unas claves ad-hoc a dicha votación.
- La urna, encargada de recibir los votos, emitir justificantes de votos (si fuese necesario), recontarlos, hacer público los resultados y una lista de comprobación de votos (si también fuese necesario).

ISBN: 978-84-9717-323-0

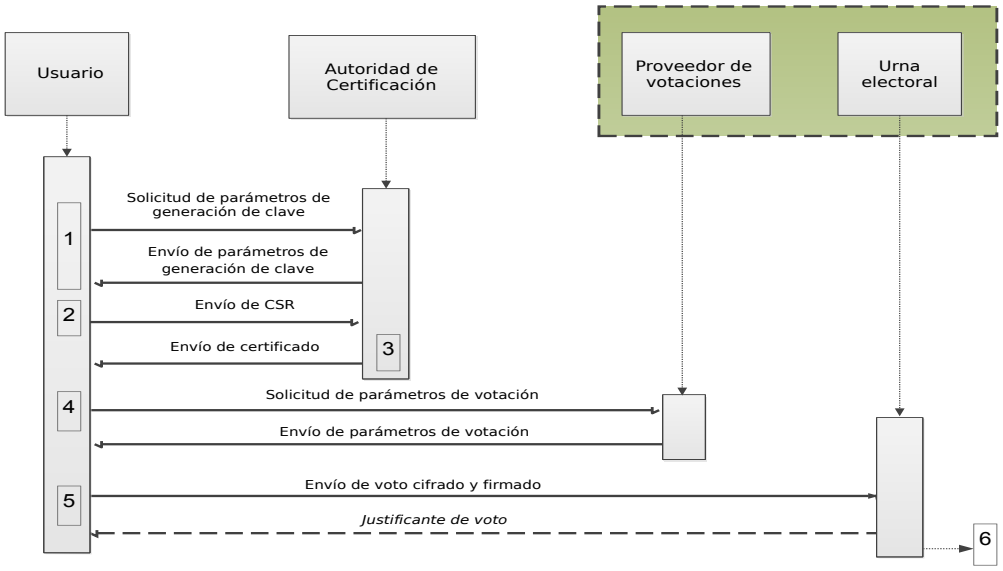


Figura 1. Proceso de votación

Contando con estos actores el proceso de votación es el siguiente (figura 1):

1. El votante se conecta con la Autoridad de Certificación y se descarga los parámetros necesarios ($u$, $n$, $\lambda$, $\mu$) para generar las claves que empleará en una determinada votación.
2. El votante, desde su terminal, genera un par de claves (pública/privada), genera un CSR con su clave pública y sus datos identificativos y se lo envía, acompañado de unas credenciales de identidad previamente definidas, a la AC.
3. La AC verifica la identidad, genera el certificado de clave pública del usuario, y almacena el certificado (registro del votante) ya que será necesario para el cálculo de uno de los parámetros de la votación. Finalmente envía el certificado de clave pública al usuario, que lo almacenará en formato PKCS12 junto con su clave privada.
4. Una vez comenzado el periodo de votación, y con los parámetros creados por la urna, el votante los descarga. Entre ellos está el conjunto de las claves públicas de todos los participantes en la votación.
5. Una vez obtenidos los parámetros, cada usuario vota, cifrando el voto con la clave pública de la urna y firmándolo en anillo con las claves asociadas al certificado emitido para él. En caso de ser necesario, recibe de la urna el justificante del voto.
6. La urna recuenta los votos y los hace públicos (en caso de no emitir recibos) o hace públicos los resultados y las etiquetas asociadas a los votantes (sin mostrar relación alguna entre ellos).

### *II-A. Parámetros para la generación de claves*

Una vez establecido cuál es el escenario de votaciones, veamos cuáles son los parámetros necesarios para la creación de las claves asociadas a una votación determinada. Es decir, para la implementación del paso 2 a partir de los parámetros que el proveedor de votaciones envía al usuario en el paso 1.

Según lo establecido en [5] el usuario ha obtenido de la AC los parámetros asociados a una votación: $u$, $n$, $\lambda$ y $\mu$. El parámetro $\lambda$ mide la seguridad del sistema a través del tamaño de $n$, siendo $n = pq = (2p' + 1)(2q' + 1)$, con $p, q, p', q'$ primos y $p', q' > 2^{\frac{\lambda}{2}}$, siendo $u$, un residuo cuadrático no trivial modulo $n$. Además $|e_1 - 2^l|, |e_2 - 2^l| < 2^\mu$ , donde $\lambda$ y $\mu$ se eligen para evitar ataques de coalición [6] y bastaría con tomar $l \approx \lambda/2$ y $\mu$ lo suficientemente pequeño como se establece en [5] para evitarlos. En nuestro caso tomamos $\mu = l - 2$ siguiendo la recomendación de [9].

Si bien la generación de $n$, a través de $p, q, p'$ y $q'$ no es inmediata, se trata de un caso específico de generación de primos robustos [7], que está bastante estandarizado y queda fuera del alcance de este artículo.

### *II-B. Generación de claves de usuario*

Llamaremos al algoritmo de generación de claves de usuario Key_gen($u$, $n$, $\lambda$, $\mu$) = $\{e_1, e_2, x = 2e_1e_2 + 1\}$, siendo los tres valores $e_1, e_2$ y $x$ primos. Cada par $e_1$, $e_2$ conforma una clave privada para la firma en anillo, siendo $x$ la clave pública asociada a dichos valores.

Con esta restricción se puede comprobar fácilmente que $0 \neq e_1 \neq e_2 \neq 0 \pmod 3$. Sin pérdida de generalidad tenemos que $e_1 = 1 \pmod 3$ y $e_2 = 2 \pmod 3$. Por lo tanto $x = 2 \pmod 3$, y $e_1, e_2 \in (2^{\frac{\lambda}{2}} - 2^\mu, 2^{\frac{\lambda}{2}} + 2^\mu)$. La estrategia para la búsqueda de estas claves estará condicionada por la

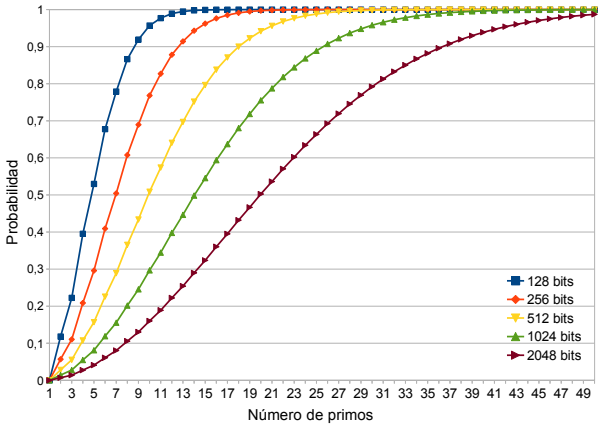


Figura 2. Número de primos para obtener un primo $2e_1e_2+1$

capacidad computacional, tanto de tiempo de procesado, como de almacenamiento en memoria.

Sin embargo, en cualquiera de los casos, la búsqueda ha de comenzar a partir de la generación de los dos primos pequeños, para posteriormente generar el grande. Por lo tanto, podemos suponer que el tiempo de comprobación de la primalidad de un número $y \in (2^{\frac{\lambda}{2}}-2^{\mu}, 2^{\frac{\lambda}{2}}+2^{\mu})$ es prácticamente constante, dependiente de $\lambda$, y que el número de comprobaciones necesarias hasta encontrar dicho número primo viene determinada por la distribución de los números primos en el intervalo. Esa distribución nos dará una función de la probabilidad de que al elegir un número entero aleatorio con distribución uniforme en ese intervalo, dicho número sea primo y lo denotaremos con $p_{\lambda,\mu}$.

Por otro lado una vez determinado que $y$ es primo sabemos que $P$(y = 1 (mod 3)) = 1/2 = $P$(y = 2 (mod 3)), dada la distribución de los números primos en sucesiones aritméticas. Con esto y teniendo en cuenta el teorema del número primo, aproximaremos la esperanza de intentos para encontrar un primo $y$ como ($1/p_{\lambda,\mu}$) y la de que tenga una determinada congruencia con 1 ó 2 módulo 3 será el doble de dicho valor, ($2/p_{\lambda,\mu}$).

Antes de comenzar nuestro análisis daremos una cota superior de una probabilidad que nos será muy útil. Sería interesante conocer cuál es la probabilidad $p$ de que a partir de un número $e_1$ dado, encontrar otro número primo $x = 2ke_1+1$, con $k$ variando en el intervalo ($2^{\frac{\lambda}{2}}$-$2^{\mu}$, $2^{\frac{\lambda}{2}}$+$2^{\mu}$), con lo que $x \in (1+2^{\lambda+1}+2^{2\mu+1}-2^{\frac{\lambda}{2}+\mu+2}, 1+2^{\lambda+1}+2^{2\mu+1}+2^{\frac{\lambda}{2}+\mu+2})$.

Utilizando la notación del teorema de Brun-Titchmarsh [8], la expresión $\prod(a,b,c)$ denota la cantidad de números primos menores que $a$ en la sucesión $x_n = c+nb$. De este teorema podemos deducir que la densidad de números primos en esa sucesión, dentro del intervalo ($2^{\frac{\lambda}{2}}$-$2^{\mu}$, $2^{\frac{\lambda}{2}}$+$2^{\mu}$), va a estar acotada por:

$$\prod(1+2^{\lambda+1}+2^{2\mu+1}+2^{\frac{\lambda}{2}+\mu+2},2e_1,1)-\prod(1+2^{\lambda+1}+2^{2\mu+1}-2^{\frac{\lambda}{2}+\mu+2},2e_1,1) < \frac{2\cdot 2^{\frac{\lambda}{2}+\mu+3}}{\varphi(2e_1)log(\frac{2^{\frac{\lambda}{2}+\mu+3}}{2e_1})} = \frac{2^{\frac{\lambda}{2}+\mu+4}}{(e_1-1)log(\frac{2^{\frac{\lambda}{2}+\mu+2}}{e_1})};$$

Por lo tanto, la probabilidad buscada $p$ estará acotada por: $p = \frac{\prod(1+2^{\lambda+1}+2^{2\mu+1}+2^{\frac{\lambda}{2}+\mu+2},2e_1,1)-\prod(1+2^{\lambda+1}+2^{2\mu+1}-2^{\frac{\lambda}{2}+\mu+2},2e_1,1)}{2^{\frac{\lambda}{2}+\mu+3}/2e_1}$

$$< \frac{\frac{2^{\frac{\lambda}{2}+\mu+4}}{(e_1-1)log\left(\frac{2^{\frac{\lambda}{2}+\mu+2}}{e_1}\right)}}{\frac{2^{\frac{\lambda}{2}+\mu+2}}{e_1}} = \frac{4e_1}{(e_1-1)log\left(\frac{2^{\frac{\lambda}{2}+\mu+2}}{e_1}\right)} \approx \frac{4}{log\left(\frac{2^{\frac{\lambda}{2}+\mu+2}}{e_1}\right)} \leq \frac{4}{log\left(\frac{2^{\frac{\lambda}{2}+\mu+2}}{2^{\frac{\lambda}{2}+2^{\mu}}}\right)}$$

## III. Estrategias de minimización

### III-A. *Estrategias de minimización de memoria*

En una estrategia de minimización de memoria, deberíamos fijar $e_i$ con $i \in \{1,2\}$ que lo conseguiremos tras un número medio de intentos ($1/p_{\lambda,\mu}$). Ahora deberíamos buscar primos $e_j$ con $i \neq j \in \{1,2\}$ lo cual conseguiremos tras ($2/p_{\lambda,\mu}$). Si $x = 2e_1e_2+1$ es primo, ya habríamos conseguido nuestro objetivo, en caso contrario buscaríamos otro $e_j$. En resumidas cuentas, estaríamos buscando números primos en la sucesión aritmética $x_{ik} = 1+2e_ik$. Aunque el número $k$ ha de ser primo y por lo tanto su distribución deja de ser uniforme, consideraremos (perdiendo una mínima precisión) que sí lo es ya que previamente hemos calculado la cota de probabilidad $p$ sobre una distribución uniforme (cota que también será válida para esta otra distribución). En ese caso, el número de $k's$ necesarias, será mayor que la esperanza de la distribución geométrica con probabilidad de éxito, $p$, es decir $1/p$. El algoritmo sería el siguiente:

Key_gen($u$, $n$, $\lambda$, $\mu$):

1. $e_i = Rand(\lambda,\mu)$
2. WHILE ($e_i$ no primo) $\{e_i = Rand(\lambda,\mu)\}$
3. $e_j = 1$, $x = 2e_ie_j$
4. WHILE ($x$ no primo)
   $\{e_j = Rand(\lambda,\mu)$
   WHILE ($e_j$ no primo) $\{e_j = Rand(\lambda,\mu)\}$
   $x = 2e_1e_2+1\}$
5. RETURN $(e_1,e_2,x)$

Teniendo en cuenta que para el cálculo del tiempo de procesado, la tarea que más tiempo requiere es calcular la primalidad de $x$, podemos afirmar que el tiempo de procesado es del orden de $1/p$. La cantidad de memoria de almacenamiento empleada se mantiene constante a lo largo de todo el proceso y se limita a dos únicos primos, el primo $e_i$ que permanece fijo y los $e_j's, i \neq j$, necesarios hasta obtener un primo de la forma $2e_je_i+1$.

### III-B. *Estrategias de minimización de tiempo de procesado*

Si por el contrario lo que queremos es seguir una estrategia de optimización del tiempo de procesado, tendremos que minimizar el número de test de primalidad para números del tamaño de $x$. Analizando nuestro objetivo podemos apreciar que el número sobre el que queremos aplicar los test

de primalidad tiene en realidad dos fuentes de aleatoriedad prácticamente independientes $e_1$ y $e_2$, por lo que quizás sería conveniente pensar en intentar usar la paradoja del cumpleaños en nuestro beneficio.

Para ello definiremos inicialmente 2 conjuntos: $P_1$ y $P_2$, donde $P_i = \{x \in (2^{\frac{\lambda}{2}} - 2^{\mu}, 2^{\frac{\lambda}{2}} + 2^{\mu}) \backslash x$ es primo y $x = i(mod\ 3)\}$. Se generan números primos en el intervalo $(2^{\frac{\lambda}{2}} - 2^{\mu}, 2^{\frac{\lambda}{2}} + 2^{\mu})$ con una semilla aleatoria con distribución uniforme. Cada vez que consigamos un primo, $r$, veremos a qué conjunto $P_i$ pertenece e iremos comprobando la primalidad de $x = 2re_j+1$, donde $e_j$ va recorriendo todos los valores de $P_j$, $(j = -i(mod\ 3))$. Si no se consigue ningún resultado positivo, añadimos $r$ al conjunto $P_i$ e iniciamos la búsqueda de un nuevo $r$. El algoritmo sería el siguiente:
Key_gen($u$, $n$, $\lambda$, $\mu$):

1. $P_1 = \emptyset$, $P_2 = \emptyset$; $i_1 = 0, i_2 = 0$
2. WHILE ($P_1 = \emptyset$ OR $P_2 = \emptyset$)
   $\{r = Rand(\lambda, \mu)$
   WHILE ($r$ no primo) $\{r = Rand(\lambda, \mu)\}$
   $k = r\ mod(3)$
   $P_k[i_k] = r$
   $i_k + +\}$
3. Éxito = False; $i = 0$
4. WHILE (!Éxito AND $i < \#P_{-k}$)
   $\{x = 1 + 2rP_{-k}[i]$
   IF ($x$ primo) {Éxito =TRUE}
   $i + +\ \}$
5. WHILE (!Éxito)
   $\{r = Rand(\lambda, \mu)$
   WHILE ($r$ no primo) $\{r = Rand(\lambda, \mu)\}$
   $k = r\ mod(3)$
   $i = 0$
   WHILE (!Éxito AND $i < \#P_{-k}$)
   $\{x = 1 + 2rP_{-k}[i]$
   IF ($x$ primo) {Éxito =TRUE}
   IF (!Éxito) {i++}}
   IF (!Éxito) $\{P_k[i] = r\}\ \}$
6. RETURN $(r, P_{-k}[i], x)$

En este caso si consideramos que $P_1$ y $P_2$ están permanentemente equilibrados y llamamos $k$ al número de $r's$ calculadas obtendremos que para un entero $m$:
$P(k \leq m) = 1 - P(k > m) = 1 - q^{\left(\frac{m-1}{2}\right)^2}$, donde hemos definido $q$ = 1 - $p$.

Calcular la esperanza analítica de esta distribución, y por tanto de la memoria necesaria, no resulta sencillo. La memoria necesaria aumentará conforme más primos $e_1$ y $e_2$ sean necesarios para el cálculo del primo $2e_1e_2 + 1$. A priori no se conoce el total de memoria necesaria pero sí que se puede realizar una cierta estimación basándonos en los resultados teóricos obtenidos, figura 2. En la Sección IV hacemos una comparativa sobre las gráficas teóricas resultantes de los valores de $p$ para diferentes valores de $\lambda$ y exponemos los resultados empíricos para el cálculo de claves reales con las características exigidas.

## IV. Resultados

En la figura 2 podemos apreciar cuál sería la curva teórica para la distribución de la probabilidad del número de intentos necesarios para el cálculo de claves en el caso de minimización de tiempo de procesado. Podemos ver cómo el número de primos que debemos extraer será mayor conforme aumente la longitud de las claves que queremos calcular.

En la tabla I indicamos los resultados obtenidos después de realizar el cálculo de 500 claves de cada tipo: número medio de primos, $e_1$ y $e_2$ calculados; el número de intentos que se han realizado antes de obtener el primo buscado; y el tiempo que se ha tardado en obtenerlo. En las dos últimas columnas indicamos el valor teórico con el que se supera el umbral del 0.5 de probabilidad de obtener un primo $2e_1e_2 + 1$ y su probabilidad asociada. Los valores de $\#P_1$ y $\#P_2$ no están perfectamente equilibrados pero la mayor diferencia entre las medias de $e_1$ y $e_2$ calculados es menor del 3 %.

En la tabla II se muestran los resultados para la estrategia de optimización de memoria. En este caso se muestran los valores de la esperanza de este sistema para obtener las claves, $(1/p_{\lambda\mu})$. Si comparamos el valor del número de extracciones esperadas para la obtención de un primo $2e_1e_2+1$, vemos que la diferencia con la estrategia de minimización de tiempo es más que considerable y además esta diferencia se incrementa conforme mayor es el tamaño de la clave buscada, llegando a ser la esperanza del número de intentos de esta estrategia mas de siete veces mayor que en la estrategia de minimización del tiempo de procesado.

## V. Conclusiones

En el artículo se muestra el proceso de generación de claves para un protocolo de votación electrónica. Se han mostrado dos estrategias distintas de obtención de claves, minimización de memoria y minimización de tiempo de procesado, y se ha

Tabla I
Resultados con minimización de tiempo

| **Bits** | $\#P_1$ | $\#P_2$ | **# x fallidas** | **Tiempo(ms)** | $m$ | **P($m$)** |
|---|---|---|---|---|---|---|
| 128 | 5,55 | 5,62 | 33 | 5,5 | 5 | 0,52976 |
| 256 | 7,70 | 7,48 | 65 | 30,5 | 7 | 0,50432 |
| 512 | 10,40 | 10,64 | 126 | 262,6 | 10 | 0,50875 |
| 1024 | 14,21 | 14,64 | 251 | 3319,9 | 15 | 0,54542 |
| 2048 | 21,23 | 21,37 | 544 | 56664,1 | 20 | 0,50248 |

Tabla II
Probabilidad y esperanza para minimización de memoria

| **Bits** | **E[nº intentos]** |
|---|---|
| 128 | 8,46254 |
| 256 | 17,6031 |
| 512 | 35,6735 |
| 1024 | 71,5329 |
| 2048 | 143,744 |

hecho un desarrollo teórico del coste de obtención de dichas claves con las dos estrategias.

Podemos concluir que ambas estrategias son válidas para el efecto que se diseñaron. Sin embargo, podemos apreciar que pensar en una estrategia de minimización de memoria exige una cantidad y tiempo de procesado que es difícil de asumir en los dispositivos actuales ya que su coste (ya sea tiempo o precio del procesador) parece resultar más caro que la memoria necesaria para acelerar el proceso.

## AGRADECIMIENTOS

Este trabajo ha sido parcialmente financiado por el proyecto CPUFLIPI (MICINN TIN2010-17298) del Gobierno de España y por la Cátedra Telefónica-Universidad de Zaragoza.

## REFERENCIAS

[1] J. Gerlach, U. Gasser, "Three Case Studies from Switzerland: E-Voting" *Berkman Center Research Publication No. 2009-03.1*

[2] U. Madise, P. Vinkel, "Constitutionality of Remote Internet Voting: The Estonian Perspective," *Juridica International XVIII*. 2011, pp. 4-16

[3] J.L. Salazar, J. Piles, J. Ruiz, J.M. Moreno-Jiménez, "Security approaches in e-cognocracy," *Computer Standards and Interfaces, vol. 32*, 2010 pp. 256–265.

[4] R.L. Rivest, A. Shamir and Y. Tauman, "How to leak a secret," *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology (ASIACRYPT '01)*, 2001, pp. 552–565.

[5] P. P. Tsang, V. K. Wei, "Short linkable ring signatures for e-voting, e-cash and attestation," *Proceedings of the First international conference on Information Security Practice and Experience (ISPEC'05)*, 2005, pp. 48–60.

[6] Camenisch, J., Lysyanskaya, A., "A signature scheme with efficient protocols," Cimato, S., Galdi, C., Persiano, G. (eds.)*Security Communication Networks, vol. 2576*, 2002, pp. 268-289

[7] J. Gordon, "Strong Primes are Easy to Find," *Advances in Criptology - Eurocrypt'84.*, 1984, pp. 216–223

[8] J. Friedlander, H. Iwaniec, 'Applications to Linear Sequences (57)," American Mathematical Society Colloquium Publications. American Mathematical Society, Providence, RI, 2010.

[9] G. Ateniese, J. Camenischm M. Joye, G. Tsudik "A Practical and Provably Secure Coalition-Resistant Group Signature Scheme," CRYPTO 2000. LNCS, vol. 1880, pp. 255–270. Springer, Heidelberg (2000)