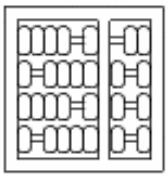

INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Verificação Eficiente de Validade de Certificados

Alexandre Almeida Lima
Ricardo Dahab

Technical Report - IC-03-025 - Relatório Técnico

November - 2003 - Novembro

VERIFICAÇÃO EFICIENTE DE VALIDADE DE CERTIFICADOS

Alexandre Lima Ricardo Dahab

RESUMO

Este artigo propõe uma alternativa ao tradicional esquema de divulgação de informações de revogação "on-line" de certificados, chamado OCSP (Online Certificate Status Protocol) [1], que utiliza apenas operações de hash e criptografia simétrica, mantendo o mesmo nível de segurança. As operações assimétricas, ou de chaves públicas, utilizadas nos OCSP-responders são muito custosas, em tempo e memória, a ponto de tornar a divulgação de informações de revogação a maior consumidora de recursos na manutenção de uma Infra-estrutura de Chaves Públicas (ICP, ou PKI--Public Key Infrastructure) [2]. Nosso esquema permite grande economia em processamento, transmissão e armazenamento, beneficiando principalmente os dispositivos móveis, sem exclusão de demais tipos de clientes.

ABSTRACT

This article proposes an alternative to the traditional OCSP (Online Certificate Status Protocol) [1] scheme of broadcasting certificates' revocation information, which uses only hash and symmetric cryptography operations, while keeping the same security level. Asymmetric, or public-key operations, used in OCSP-responders are very costly, both in time and memory requirements, which makes certificate revocation management the largest resource-consuming activity in a Public-Key Infrastructure (PKI) [2]. Our proposed scheme provides a substantial reduction in processing, bandwidth and storage requirements, making it especially suitable for mobile clients.

1 INTRODUÇÃO

Diversos motivos podem levar à revogação de um certificado. O comprometimento da chave privada, a mudança de status do proprietário, dentre outros, são alguns motivos que levam uma PKI a publicar uma lista dos certificados emitidos por ela e que foram revogados. A confecção e distribuição destas informações de revogação são o principal custo de manutenção de uma PKI [3].

Baseando-nos nestes fatos, propomos um esquema que utiliza chaves secretas compartilhadas entre determinadas Autoridades Certificadoras (AC, ou CA – Certification Authority) de uma PKI, facilitando a verificação e o envio de provas de validade de certificados.

Por meio de um processo que funciona de maneira similar ao OCSP (On-line Certificate Status Protocol) [1], mas sem utilização de criptografia assimétrica, pretende-se beneficiar principalmente os clientes móveis que possuem poucos recursos de processamento, memória e comunicação. Os ganhos em tempo e memória estão espalhados por todo o

processo de validação e compensam grandemente o custo adicional imposto por nosso esquema.

A maior parcela do nosso custo está concentrada no processamento feito no OCSP-*responder* (que chamaremos daqui a diante de OCSP-R) ligado à CA âncora de confiança (que passaremos a chamar de CA-Anc) onde as operações consomem um tempo da ordem de dezenas de microssegundos, o que é pouco quando comparado a uma verificação de assinatura de cerca de duas ordens de grandeza maiores.

No OCSP tradicional a CA produz, para cada certificado previamente emitido, um texto assinado com o número de série do certificado, o status do certificado e o período de validade da informação de revogação. Em seguida envia esses textos para o OCSP-R, que é o responsável pela divulgação destas informações de revogação. Chamamos a esse processo de **carga do OCSP-R**, ou simplesmente **carga**.

Os verificadores de certificados enviam o número de série dos certificados que desejam verificar ao OCSP-R, e recebem deste o respectivo texto assinado pela CA emissora. Isso não exige que o verificador tenha qualquer relação de confiança com o OCSP-R.

No nosso esquema, tanto no processo de carga como no envio e verificação de tais assinaturas, os custos ficam reduzidos a um mínimo de processamento e transmissão. Também teremos a diminuição tanto no tamanho das mensagens trocadas entre verificador e OCSP-Rs como na quantidade de memória utilizada para guardar as informações de revogação.

Este trabalho está dividido da seguinte forma: na próxima seção apresentamos uma visão geral do funcionamento do esquema; em seguida, descrevemos a implementação do esquema nas PKIs atuais; a Seção 4 apresenta o processo de atualização periódica das tabelas dos OCSP-Rs; na seção 5, mostramos o processo de verificação de um certificado; na Seção 6 exibimos um exemplo simples; a Seção 7 apresenta as comparações entre o sistema proposto e o esquema OCSP tradicional; a Seção 8 conclui o artigo.

2 VISÃO GERAL

As principais entidades que participam deste esquema são: o cliente, que deseja verificar a validade de um certificado enviado por um provedor; o provedor que deseja fornecer algum serviço autenticado ao cliente; e o verificador responsável por conseguir uma prova de validade do certificado do provedor. O verificador pode estar alojado no cliente, no próprio provedor, ou ser uma entidade independente, isto depende das capacidades computacionais e de transmissão das entidades envolvidas. Por exemplo, atualmente um dispositivo móvel é pouco provável que tenha recursos suficientes para desempenhar as tarefas necessárias a um verificador. Para simplificar vamos imaginar o verificador como uma entidade independente.

A figura central do esquema proposto é o verificador, que inicialmente recebe de um cliente o certificado de um provedor a ser verificado. O verificador consulta os Diretórios (Dir), responsáveis por divulgar certificados, das CAs pertencentes ao caminho de certificação e obtém assim informações tais como: endereços dos OCSP-Rs, número de série, CA emissora, distinguished name (DN), etc.

Já as informações de revogação são obtidas de tal forma que, ao final do processo, o verificador repasse ao cliente um “carimbo” de tamanho reduzido e impossível de ser

forjado que garante o atual estado do caminho de certificação. Este carimbo é emitido pelo OCSP-R da CA âncora de confiança do cliente (OCSP-Anc).

Fig. 1 - Resumo do Esquema Proposto.

Conforme Fig. 1, o verificador trocará registros com os OCSP-Rs do caminho (OCSP-Pai, OCSP-Int e OCSP-Rz) e cada OCSP-R repassará ao verificador um "Recibo". Após percorrer o caminho de certificação, o verificador terá uma coleção de recibos, e entregará tudo ao OCSP-Anc que utilizando operações hashes e simétricas e o esquema proposto, terá a garantia de que todas as CAs do caminho de certificação foram consultadas pelo verificador e que os estados recebidos não foram alterados por alguma entidade maliciosa.

A confecção do "Carimbo" é feita utilizando, dentre outros valores, um segredo compartilhado entre o cliente, a sua CA âncora, e o OCSP-Anc. Isto garante ao usuário que o carimbo não foi forjado por ninguém.

3 DUAS NOVAS TABELAS

Inicialmente é importante ressaltar que as dimensões das funções criptográficas utilizadas como modelo neste esquema acompanham as recomendações do NIST quanto à segurança dos algoritmos criptográficos [4,5], são elas: função hash de 32 bytes (256 bits) de valor hash, e algoritmo simétrico com 24 bytes (192 bits) de chave privada.

Para primeira tabela, propomos que a CA-Anc mantenha uma tabela (**TabPath**) com um registro para cada CA descendente (CA-Desc), cujo conteúdo serão apenas dois campos: o valor hash do DN da CA-Desc (único para cada CA); e um valor secreto (32 bytes) fornecido à CA-Anc pela própria CA-Desc no momento de sua certificação. Chamaremos este valor de *PathID* (identificador de caminho).

Repare que o custo de atualização desta tabela não é alto, pois como se tratam de CAs, seus valores são atualizados com pouca freqüência.

Por fim, quando da criação de um novo certificado pela CA-Desc, esta envia um outro valor secreto (32 bytes) para o cliente, que no futuro o ajudará a confirmar a procedência de uma prova de validade de certificado (*Carimbo*). Chamaremos este valor de *PathIDCli*, ele é calculado conforme (1), onde *HDNCli* é o valor hash do DN do cliente.

$$PathIDCli \quad Hash(PathID \,||\, HDNCli) \tag{1}$$

Faremos bastante uso do valor hash do DN de uma entidade (HDNEntidade), pois entendemos que esta seja uma forma compacta (32 bytes) de identificar univocamente uma entidade. Caso necessário, para garantir unicidade, podemos incluir a chave pública de Entidade como parâmetro adicional a ser concatenado ao DN.

A segunda tabela, **TabChave**, diz respeito ao relacionamento entre CAs pai e filha. Quando uma CA-Pai emite um certificado para uma CA-Filha, elas estabelecem um valor

secreto K^0 (32 bytes), que determinará, durante o tempo de validade do certificado da CA-Filha, os valores da chave compartilhada pelas duas CAs em cada período de divulgação de informações de revogação.

Por simplicidade, vamos supor, que os certificados possuam validade de um ano e que as informações de revogação tenham periodicidade de um dia. Portanto o valor da chave entre a CA-Pai e a CA-Filha no i-ésimo dia após a emissão do certificado da CA-Filha vale: $K^i \quad Hash^{365\ i}(K^0)$.

Uma CA manterá uma tabela TabChave com um registro para cada CA-Filha (haverá também entradas destinadas à sua CA-Pai e às CAs com certificações cruzadas) e tais registros possuem os seguintes campos: valor hash do DN da CA-Filha; data de emissão do certificado da CA-Filha; e o valor K^0_{Filha}. Como não se espera que o número de CAs-Filha seja grande e a atualização de seus certificados não seja freqüente, a manutenção da tabela também não é cara.

4 INTERAÇÃO CA ↔ OCSP-R (ATUALIZAÇÃO)

Suponha uma CA, chamada CA-Inter (de intermediária), cujo certificado foi emitido por uma CA chamada CA-Pai. No momento da emissão do certificado as duas CAs estabelecem uma chave privada de 32 bytes que iremos chamar de Key^0_{Pai}. Vamos supor ainda que esta CA-Inter emita um certificado para outra CA, chamada de CA-Filha, e estabeleçam como chave privada a chave Key^0_{Filha}. Ambas as chaves constam da tabela TabChave mantida pela CA-Inter, também há referência para a chave Key^0_{Pai} na tabela TabChave da CA-Pai, assim como para a chave Key^0_{Filha} na CA-Filha.

Estas chaves serão as sementes para a cadeia de hash que determinará a chave válida para um respectivo dia. Por exemplo, a chave válida entre a CA-Inter e a CA-Pai no i-ésimo dia após a emissão do certificado da CA-Inter será:

$$Key^i_{Pai} \quad hash^{365\ i}(Key^0_{Pai}).$$

De posse destas chaves, as CAs atualizam diariamente os seus OCSP-Rs com duas tabelas, uma para os certificados-folha (**TabFolha**) e outra para as suas CAs-Filha (**TabFilha**).

A TabFolha contém os seguintes campos:

SN	Status	
	Estado	Recibo
	OK/NOK	H(Estado||HDNFilho||K^i_{Pai})

Fig. 2 – TabFolha, atualizada com os dados dos certificados-folha.

SN = Número de Série do Certificado, possibilita a localização (única) do *Status* de um certificado pelo OCSP-R;

Status = Concatenação de dois valores. O primeiro representando o *Estado* do certificado-folha que pode ser OK ou NOK (caso o OCSP-R desconheça o certificado, ele emite um estado UNKNOWN) e o segundo é o *Recibo*, que é o valor hash calculado sobre a concatenação dos valores: *Estado*; *HDNFilho* (valor hash do DN do certificado-folha); e K^i_{Pai} (a chave do dia entre CA-Inter, emissora, e a CA-Pai).

Os campos envolvidos no Recibo têm a finalidade de garantir: a integridade do campo Estado; a originalidade do certificado que está verificando (um falsificador poderia no decorrer da verificação trocar o certificado verificado por outro); a fonte que originou o recibo; e a data da emissão do recibo. É impossível, para qualquer outra entidade, emitir um Recibo válido, pois desconhece a chave K^{i}_{Pai}.

Os registros da TabFolha possuem 49 bytes de tamanho, isto supondo que o número de série dos certificados possua 16 bytes e o campo Estado tenha 1 byte.

A TabFilha, destinada às CAs-Filha, é similar à tabela acima só que possui um campo a mais.

SN	Status		Chave
	Estado	Recibo	
	OK/NOK	$H(Estado \| HDNFilho \| K^{i}_{Pai})$	$K^{i}_{Pai}[K^{j}_{Filho} \| HDNFilho]$

Fig. 3 – TabFilha, atualizada com os dados das CAs-Filha.

O campo *Chave* é responsável por levar até a CA-Anc a chave compartilhada entre esta CA (CA-Inter) e a sua CA-Filha pertencente ao caminho, válida em um determinado dia. Esta chave permitirá à CA-Anc conferir o Recibo emitido pela CA-Filha em relação ao seu dependente pertencente ao caminho de certificação, bem como a coleção dos campos Chave irá permitir a verificação da ordem de participação das CAs pertencentes ao caminho.

Esta operação é feita por meio de cifragem simétrica, utilizando a chave que relaciona a CA-Inter com a sua CA-Pai (K^{i}_{Pai}). São cifradas a chave que combinadas entre a CA-Inter e a CA-Filha pertencente ao caminho (K^{j}_{Filho}) e o respectivo valor hash do DN desta CA (*HDNFilho*).

Observe que somente a CA-Inter tem conhecimento das duas chaves e portanto somente ela pode emitir um campo Chave válido.

O número de registros em uma TabFilha não é muito grande, pois contém basicamente apenas as CAs-Filha, e cada registro possui 113 bytes. A soma das duas tabelas fica muito abaixo das tabelas dos OCSP-Rs atuais, que envolvem registros com pelo menos o número de série(16), o estado(1) e uma assinatura da CA-Inter(256), totalizando 273 bytes. Supomos aqui que o algoritmo de chave pública utilizado pelos OCSP-Rs seja o RSA-2048.

O OCSP-Anc será atualizado da mesma forma que os demais OCSP-Rs, tanto para os usuários-folha quanto para as CAs-Filha, porém, por exercer um papel diferente dos demais OCSP-Rs, ele terá duas tabelas a mais: uma tabela, **TabPathOCSP**, idêntica à TabPath armazenada na CA-Anc; e uma tabela chamada **TabChaveOCSP**, contendo as chaves válidas para o respectivo dia, entre a CA-Anc e as suas CAs-Filha.

As duas tabelas são indexadas pelos valores hash dos DN das CAs Descendentes e CAs-Filha respectivamente, tendo cada registro 64 bytes. Mais uma vez não se espera que as tabelas tenham um tamanho expressivo.

O objetivo da TabPathOCSP é possibilitar que o OCSP-Anc emita um Carimbo, personalizado para o cliente, já o objetivo da TabChaveOCSP é possibilitar que o OCSP-Anc abra os Status e Chaves montados pelas CAs intermediárias e enviados pelo verificador.

Tabela 1-Tabelas envolvidas no processo de verificação.

Nome Tabela	Local	Registro (Bytes)	Atualização
TabPath	CA-Anc	64	Esporádica
TabChave	CAs	68	Anual
TabFilha	OCSP-Rs	113	Diária
TabFolha	OCSP-Rs	49	Diária
TabPathOCSP	OCSP-Anc	64	Esporádica
TabChaveOCSP	OCSP-Anc	64	Diária

5 INTERAÇÃO VERIFICADOR OCSP-Rs (VERIFICAÇÃO)

De posse do certificado a ser verificado, o verificador vai a procura dos OCSP-Rs relativos às CAs pertencentes ao caminho de certificação. Os OCSP-Rs não são confiáveis pelo cliente, a exceção do OCSP-Anc.

O verificador busca junto aos respectivos Diretórios os certificados das CAs que compõem o caminho. Isto tornará possível ao verificador saber qual a próxima CA do caminho e onde encontrar seu OCSP-R.

A comunicação entre verificador e os OCSP-Rs é feita usando a mesma forma de consulta/resposta dos OCSP-Rs atuais, porém o que é trocado entre eles é um registro com os seguintes campos:

Fig. 4 - Registro trocado entre verificador e OCSP-Rs.

Tipo = Determina o tipo de registro (0 = *Consulta*; 1 = *Resposta*; 2 = *Seqüência*; 3 = *Carimbo*);

Onde *Consulta* indica um registro enviado pelo verificador a um OCSP-R; *Resposta* indica que é a resposta a uma consulta, enviada pelo OCSP-R ao verificador; *Seqüência* é o registro enviado pelo verificador ao OCSP-Anc contendo a seqüência de Status e Chaves colhidos pelo caminho; e Carimbo é um registro contendo dados confirmando o Estado do certificado pela CA-Anc.

SNult = Número de série do certificado descendente direto da CA consultada;

Como as tabelas dos OCSP-Rs estão indexadas pelos números de série dos certificados, este campo é suficiente para que o OCSP-R localize nas tabelas TabFolha ou TabFilha as informações do respectivo certificado.

Status = Traz o estado do certificado e um recibo confirmando a origem do dado;

Vale ressaltar que HDNFilho se refere à entidade descendente direta (filho) da CA consultada.

Chave = Propaga a chave utilizada na confecção do recibo.

O registro enviado pelo verificador aos OCSP-Rs intermediários, possui apenas os dois primeiros campos, totalizando 17 bytes. O registro de retorno dos OCSP-Rs ao verificador possui 114 bytes.

O processo em linhas gerais faz com que o verificador percorra todo o caminho de certificação, colecionando Status e Chaves, de modo que ao se dirigir ao OCSP-Anc

forneça estes dados para que este possa executar uma conferência e emitir o "Carimbo" confirmando ou não a validade do caminho de certificação.

O verificador ao identificar que o próximo OCSP-R a ser consultado é o OCSP-Anc, deve encaminhar um registro com o campo *Tipo* valendo 3. Este registro traz todos os Status e Chaves recolhidas até o momento, em ordem inversa à obtida pelo verificador.

O registro enviado pelo verificador ao OCSP-Anc possui os seguintes campos:

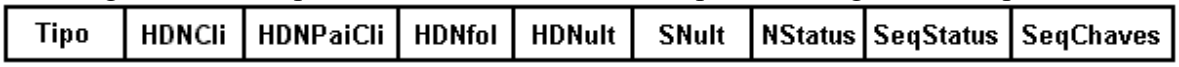

Fig. 5 - Registro enviado pelo verificador ao OCSP-Anc.

HDNCli = Hash do DN do certificado do Cliente que vai receber a prova de validade (Carimbo);

Este campo é necessário para que o OCSP-Anc possa confeccionar o carimbo personalizado para o cliente.

HDNPaiCli = Hash do DN da CA emissora do certificado do cliente;

O objetivo deste campo é possibilitar que o OCSP-Anc localize na TabPathOCSP, o respectivo PathID, para poder confeccionar o carimbo.

HDNfol = Hash do DN do certificado-folha que se está verificando o caminho;

Este campo garante que o certificado-folha verificado não seja trocado durante o processo. O registro retornado pelo OCSP-R da CA-Pai (emissora do certificado do provedor), contém o campo Recibo calculado utilizando este HDNfol.

HDNult = Hash do DN da última entidade consultada no caminho;

Indica qual foi a última entidade, pertencente ao caminho, que foi consultada quanto à validade do certificado (esta deve ser uma CA-Filha ou um cliente-folha descendente direto da CA-Anc).

SNult = Número de série do certificado referenciado em HDNult;

Serve para que a CA-Anc possa verificar o Estado do certificado da última CA que participou do processo e que é sua descendente direta (filha).

NStatus = Número de Status que serão transmitidos;

SeqStatus = Seqüência de Status (iniciando pelo último obtido pelo verificador);

SeqChaves = Seqüência de Chaves (na mesma ordem de SeqStatus);

Em um caminho de tamanho l, o número de Status é $l - 1$ e o de Chaves é $l - 2$. Portanto teremos um total de $97 \cdot l - 15$ bytes enviados pelo verificador ao OCSP-Anc.

A verificação no OCSP-Anc é feita da seguinte forma: inicialmente o OCSP-R utilizando o SNult e consultando sua TabFilha, verifica o Estado do certificado da última CA intermediária no caminho; depois obtém a chave do dia que o relaciona à esta CA (K_{Pai}^{i}), usando HDNult e consultando a TabChaveOCSP. De posse desta chave o OCSP-Anc decifra a primeira Chave da seqüência, obtendo HDNFilho. Podendo agora verificar o primeiro Status recebido na seqüência, calculando $Hash(Estado \,||\, HDNFilho \,||\, K_{Pai}^{i})$.

Com o deciframento do primeiro campo Chave obtém-se também a próxima chave (K_{Filho}^{j}) que permitirá o deciframento do próximo campo Chave assim como a conferência do próximo Status.

Este procedimento segue até que finalmente é possível conferir o último Status e o OCSP-Anc emitir um "Carimbo" certificando que o estado geral (*EstGeral*) do caminho está OK ou NOK.

Este carimbo é confeccionado da seguinte forma: primeiro busca-se em TabPathOCSP, utilizando o HDNPaiCli, o valor PathID e então se calcula o PathIDCli conforme (2); depois calcula o hash da concatenação do PathIDCli com a identificação do certificado que foi verificado (HDNFol), o resultado da verificação (EstGeral) e a *Data* (somente dia/mês/ano), conforme (3).

$$PathIDCli \quad Hash(PathID \| HDNCli) \tag{2}$$

$$Carimbo \quad Hash(PathIDCli \| HDNFol \| EstGeral \| Data) \tag{3}$$

Finalmente o OCSP-Anc retorna ao verificador o Carimbo e o EstGeral (diz se o estado geral de todo o caminho de certificação está OK ou não) em um registro com 66 bytes, como prova da validade (ou não) do certificado.

Tipo	HDNfol	EstGeral	Carimbo

Fig. 6 - Registro enviado pelo OCSP-Anc ao verificador.

O verificador retorna ao cliente um registro com os seguintes campos: HDNfol , EstGeral, e o Carimbo. O cliente calcula então Carimbo utilizando o seu PathIDCli e caso coincidam, é confirmado que o caminho de certificação do referido HDNfol, foi todo verificado e o EstGeral representa o estado atual do certificado.

6 UM EXEMPLO

Vamos usar a seguinte árvore de PKI, onde o cliente (Cli) possui a CA-Anc como sua âncora de confiança, e o provedor (Prv) está subordinado a uma PKI que possui a CA-Rz como sua CA raiz. Para generalizar o exemplo, supomos que existe uma certificação cruzada entre a CA-Anc e a CA-Rz (conforme Fig 7).

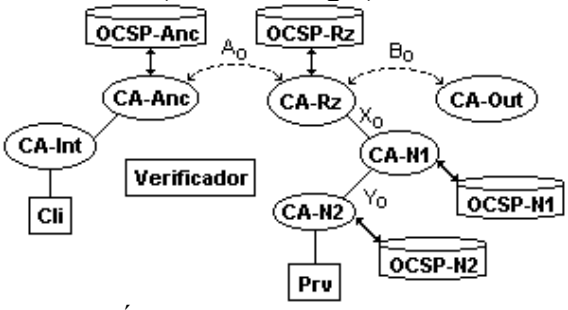

Fig. 7 - Árvore da PKI utilizada no exemplo.

Cada CA possui seus respectivos Diretório e OCSP-R responsáveis por divulgar o próprio certificado e as provas de validade dos certificados emitidos por esta CA. Também temos as chaves compartilhadas entre as CAs pai e filhas, representada pelas letras com índice 0.

Inicialmente o verificador (Ver) envia um registro ao OCSP-R da CA-N2 (N2) com campo Tipo indicando uma consulta (Tipo = 0). O SNult determina o certificado consultado. O símbolo ⌶ indica campo em branco.

Na mensagem 02, o campo Tipo é preenchido com o valor 1 indicando que é uma resposta do OCSP-R ao verificador. Com a intenção de facilitar o trabalho do verificador, o campo SNult já é preenchido com o número de série do certificado da CA-N2 emitido pela

CA-N1, porém o verificador já possui esta informação ao ter verificado anteriormente o caminho de certificação e seus certificados.

Tabela 2-Mensagens entre verificador e OCSP-Rs.

Msg	De	Para	Tipo	SNult	Status		Chave
					Estado	Recibo	
01	Ver	N2	0	2364		I	I
02	N2	Ver	1	9832	OK	OK || Prv || Y_i	I
03	Ver	N1	0	9832		I	I
04	N1	Ver	1	4288	OK	OK || CA-N2 || X_j	$X_j[Y_i \|\| CA\text{-}N2]$
05	Ver	Rz	0	4288		I	I
06	Rz	Ver	1	9143	OK	OK || CA-N1 || A_K	$A_k[X_j \|\| CA\text{-}N1]$

O OCSP-R da CA-N2 (N2) consulta sua TabFolha e busca o Status atualizado, emitido pela CA-N2, e o concatena ao registro. Como a CA-N2 é a CA emissora do certificado-folha, ela não preenche o campo Chave desta mensagem, pois não existe uma chave entre ela o provedor.

O verificador envia agora ao OCSP-R da CA-N1 (N1) um registro de consulta com os valores nos campos Tipo e SNult, conforme mostrado.

N1 preenche a resposta da mesma forma como foi feita pelo OCSP-N2, só que agora ele emite por último o campo Chave retirado da tabela TabFilha. O mesmo procedimento é feito na resposta do OCSP-R da CA-Rz (Rz) ao verificador.

Após esta troca de mensagens, o verificador envia ao OCSP-Anc o seguinte registro:

Tipo	HDNCli	HDNPaiCli	HDNFol	HDNUlt	SNUlt	NStatus	SeqStatus	SeqChaves
2	Cli	CA-Int	Prv	AC-Rz	9143	03	Status3|| Status2|| Status1	Chave3|| Chave2

Fig. 8 - Mensagem trocada entre verificador e OCSP-Anc.

Enquanto SNult permite ao OCSP-Anc verificar a validade do certificado de CA-Rz, HDNult indica qual chave (A_k) deve ser usada, tanto para verificar o primeiro Status da seqüência quanto para obter a próxima chave.

O OCSP-Anc faz as seguintes operações após verificar o certificado da CA-Rz:

1) $A_k^{-1}[A_k[X_j \| CA\quad N1]]$ e obtém X_j; e o respectivo DN da CA-N1;

2) Calcula: $Hash(Status3.Estado \| CA\quad N1 \| A_k)$ e compara com $Status3.Recibo$;

3) Calcula: $X_j^{-1}[X_j[Y_i \| CA\quad N2]]$, obtém Y_i e o respectivo DN da CA-N2;

4) Calcula: $Hash(Status2.Estado \| CA\quad N2 \| X_j)$ e compara com $Status2.Recibo$;

5) Calcula: $Hash(Status1.Estado \| Prv \| Y_i)$ e compara o resultado com $Status1.Recibo$;

6) Conforme os Estados contidos nos Status, ele determina *EstGeral* e então calcula:
$PathIDCli \quad Hash(PathID \| HDNCli)$ e emite $Carimbo \quad Hash(PathIDCli \| HDNFol \| \| EstGeral \| Data)$.

7 COMPARAÇÕES

Vamos comparar o esquema proposto ao esquema OCSP tradicional. Para isto dividimos as comparações em três áreas distintas: processamento, transmissão e armazenagem. Também utilizamos três exemplos de PKIs (pequena, média e grande) para abranger uma maior diversidade de situações.

7.1 Processamento

Para comparar os desempenhos em processamento, vamos sugerir, de acordo com trabalhos consultados [6,7,8,9,10,11], a seguinte relação entre as operações criptográficas:

Operação	Ordem	Obs:.
Hash	$O(1)$ [illegible]	Proc. 512 bits entrada.
Simétrica	$O(1)$ [illegible]	Proc. 512 bits entrada.
Verificação	$O(1)ms$	-
Assinatura	$O(10)ms$	-

Para simplificar os cálculos, as PKIs que utilizaremos como exemplo nesta avaliação de processamento, estão representadas em ordens de grandeza com as seguintes dimensões:

Descrição	PKI_{Peq}	$PKI_{Méd}$	PKI_{Grd}
Nível do Certific. (l)	*O(1)*	*O(1)*	*O(10)*
Nº Clientes-folha (CliFol)	*O(10.000)*	*O(100.000)*	*O(1.000.000)*
Nº CAs na PKI (CAs)	*O(1)*	*O(10)*	*O(100)*

7.1.1 Custo Computacional na Carga

Esquema Tradicional:

Cada CA atualiza o seu OCSP-R montando uma tabela com o número de série do certificado, o status do certificado até aquele momento e um texto assinado, cujo conteúdo são esses dados mais a data desta emissão. Assim, todas as CAs farão este cálculo (envolvendo uma assinatura) para cada um de seus clientes-folha e CAs-filha.

PKI	[O(CliFol)+O(CAs)] O(Ass.)	Resultado
Peq	*[O(10.000)+O(1)] O(10)*	*O(1.000)seg*
Méd	*[O(100.000)+O(10)] O(10)*	*O(10.000)seg*
Grd	*[O(1.000.000)+O(100)] O(10)*	*O(100.000)seg*

Esquema Proposto:

Para atualizar os OCSP-Rs cada uma das CAs monta a sua TabFolha com um registro para cada um dos seus clientes-folha (envolve o cálculo de um hash por registro) e monta a sua TabFilha com um registro para cada uma de suas CAs-filha (envolve o cálculo de um hash e uma cifra simétrica por registro). Também temos a TabChaveOCSP com um registro

para cada uma das CAs-filha, porém o cálculo das K^{i}_{Filha} $Hash^{365\ i}(K^{0})$ pode ser otimizado e realizado em horários ociosos da CA, não significando um custo no momento da carga.

Então teremos o seguinte cálculo para cada PKI:

O(CliFol) O(hash)+O(CAs) [O(hash)+O(simét)]

PKI	**Resultado**
Peq	O(100)ms
Méd	O(1)seg
Grd	O(10)seg

Significando uma economia de *4 ordens de grandeza (> 99%)* em cada tipo de PKI.

7.1.2 Custo Computacional no OCSP-Anc em uma Consulta

Esquema Tradicional:

No esquema tradicional, todos os OCSP-Rs desempenham a mesma função, ou seja, recebem do verificador uma consulta com o número de série de um certificado e montam o registro de resposta com o conteúdo anteriormente passado pela respectiva CA. O OCSP-R não tem que executar nenhum cálculo, portanto o custo computacional no OCSP-R é desprezível.

Esquema Proposto:

Os cálculos executados pelo OCSP-Anc em um caminho genérico de tamanho l são: l 1 operações hash para calcular os campos Recibos; l 2 operações de chave simétrica para recuperar as chaves; e mais duas operações hash para confeccionar o Carimbo. Portanto gastaremos: (l 1) $O(1)$ ■ (l 2) $O(1)$ ■ 2 $O(1)$ ■ $O(l)$ $O(1)$ ■. Então:

PKI	$O(l)$	$O(l)$ $O(1)$ ■
Peq	$O(1)$	$O(10)$ ■
Méd	$O(1)$	$O(10)$ ■
Grd	$O(10)$	$O(100)$ ■

Teremos *a grande parte dos custos, na ordem de dezenas de microssegundos*, o que é pequeno comparando-se aos demais ganhos em computação, transmissão e armazenagem.

7.1.3 Custo Computacional na Verificação da Validade dos Certificados em um Cliente

Esquema Tradicional:

O verificador, envia ao cliente a cadeia de respostas dos OCSP-Rs (l respostas). O cliente deve verificar a assinatura de cada resposta (são l verificações) para se certificar da origem destas respostas. Isto resulta no cálculo de l verificações de assinatura. Portanto teríamos: $O(l)$ $O(1)ms$. Tomando como base os exemplos:

PKI	$O(l)$	$O(l)$ $O(1)ms$
Peq	$O(1)$	$O(10)ms$
Méd	$O(1)$	$O(10)ms$
Grd	$O(10)$	$O(100)ms$

Podemos ter os custos destas verificações chegando à ordem das centenas de milissegundos (em uma máquina com recursos), o que seria um preço muito alto para os dispositivos móveis que, por dispor de poucos recursos, têm as ordens dos tempos nas operações criptográficas muito elevadas.

Esquema Proposto:

O verificador passa ao cliente a prova de validade do seu certificado, que se constitui dos seguintes campos: HDNfol; EstGeral; e o Carimbo. O cliente então calcula o valor do Carimbo, utilizando o seu PathIDCli. Por fim o compara com o Carimbo recebido, confirmando ou não que o certificado de HDNfol foi verificado até a sua CA-Anc. Este processo envolve apenas um cálculo de hash, ficando portanto em $O(1)$ [3].

Repare que é um tempo constante, o que é uma característica importante principalmente para dispositivos móveis onde caminhos muito extensos são proibitivos.

Quanto à economia de tempo, para as PKIs sugeridas, teríamos uma economia de *4 ordens de grandeza (> 99%)*.

7.2 Transmissão

Diferente das comparações feitas em Processamento, as transmissões podem ser aferidas com maior precisão. Portanto exemplificamos nossas PKIs com os seguintes valores.

Descrição	**PKI_{Peq}**	**$PKI_{Méd}$**	**PKI_{Grd}**
Nível do Certificado (l)	4	8	15
Nº de Clientes-folha (CliFol)	50.000	600.000	3.000.000
Nº de CAs na PKI (CAs)	8	40	100

Vamos supor que o algoritmo de chave pública utilizado pelos OCSP-Rs seja o RSA-2048 e que o tamanho de um certificado seja de 1.024 bytes.

7.2.1 Custo de Transmissão na Carga

Esquema Tradicional:

O volume de dados a ser transmitido aos OCSP-Rs será de um registro com 273 bytes (Número de Série (16) + Estado (1) + Assinatura (256)) para cada cilente-folha e CA. Portanto teremos:

PKI	(*CliFol* *CAs*) 273	**Resultado**
Peq	(50.000 + 8) 273	13.332 KB
Méd	(600.000 + 40) 273	159.971 KB
Grd	(3.000.000 + 100) 273	799.831 KB

Esquema Proposto:

O volume transmitido será: um registro de 113 bytes da TabFilha (Número de Série (16) + Estado (1) + Recibo (32) + Chave (64)) para cada CA-Filha; e um registro de 49 bytes da TabFolha (Número de Série (16) + Estado (1) + Recibo (32)) para cada cliente-folha da PKI. Portanto teremos em transmissões os seguintes gastos:

PKI	*CAs* 113 *CliFol* 49	**Resultado**
Peq	8 113 + 50.000 49	2.393 KB
Méd	40 113 + 600.000 49	28.715 KB
Grd	100 113 + 3.000.000 49	143.565 KB

Significando uma *economia de 82% de transmissão* nos três exemplos.

O OCSP-Anc, por exercer um papel diferenciado, recebe da CA-Anc a TabChaveOCSP que contém as chaves válidas para aquele dia entre a CA-Anc e as suas CAs-Filha, com cada registro tendo 64 bytes (HDN(CA-Filha) (32) + Key_{Filha} (32)), significando um acréscimo de transmissão muito pequeno (< 7 KB) a ser transmitido ao OCSP-Anc.

7.2.2 Custo de Transmissão na Interação Verificador OCSP-Rs

Esquema Tradicional:

O verificador apenas envia um registro com o Número de Série (16) e o OCSP-R retorna um registro com o Número de Série (16), o Estado (1) e a Assinatura (256), totalizando 289 bytes transmitidos em uma consulta/resposta.

Portanto num caminho de tamanho l teríamos: 289 l bytes trocados.

PKI	l	289 l
Peq	4	1.156 Bytes
Méd	8	2.312 Bytes
Grd	15	4.335 Bytes

Esquema Proposto:

O verificador troca 67 bytes com o primeiro OCSP-R do caminho, depois troca 131 bytes com os l 2 OCSP-Rs intermediários, e por fim troca 97 l 51 bytes com o OCSP-Anc. Totalizando 228 l 144 bytes trocados no processo.

PKI	l	228 l 144
Peq	4	768 Bytes
Méd	8	1.680 Bytes
Grd	15	3.276 Bytes

Teremos *economias superiores a 24% em comunicação* entre o verificador e os OCSP-Rs.

7.2.3 Custo de Transmissão na Interação Verificador Cliente

Esquema Tradicional:

O verificador enviará ao cliente as provas de validade dos certificados (l provas). Uma prova de validade de um certificado é o próprio registro que foi enviado pelo OCSP-R, com 273 bytes.

PKI	l	273 l
Peq	4	1.092 Bytes
Méd	8	2.184 Bytes
Grd	15	4.095 Bytes

Esquema Proposto:

O verificador enviará ao cliente somente 65 bytes: HDNfol(32) + EstGeral(1) + Carimbo(32).

Significando uma *economia de pelo menos 94% em transmissão*.

7.3 Armazenamento

7.3.1 Custo de Armazenamento no Verificador

Esquema Tradicional:

O verificador deve ter armazenado até o final do processo todos os certificados e provas de validade do caminho de certificação, pois enviará tudo isto ao cliente. Portanto teremos:

PKI	l	1024 l 273 l
Peq	4	5.188 Bytes
Méd	8	10.376 Bytes
Grd	15	19.455 Bytes

Esquema Proposto:

O verificador guardará todos os certificados do caminho, os l 1 Recibos (32) do caminho e os l 2 campos Chave (64) recebidos pelo caminho. Por fim, terá as informações adicionais (HDNCli(32) + HDNPaiCli(32) + HDNFol(32) + HDNUlt(32) + SNUlt(16) + NStatus(1)) que enviará ao OCSP-Anc e o Carimbo (32) que receberá deste e enviará ao cliente. Totalizando: 1.024 l (l 1) 32 (l 2) 64 145 32 = 1.120 l 17 bytes.

PKI	l	1.120 l 17
Peq	4	4.497 Bytes
Méd	8	8.977 Bytes
Grd	15	16.817 Bytes

Significando *economias de armazenamento de 13%* nos exemplos.

7.3.2 Custo de Armazenamento nos OCSP-Rs Intermediários

Esquema Tradicional:

Os OCSP-Rs guardarão registros de 273 bytes para cada cliente-folha e CA-Filha:

PKI	273 *CliFol* 273 *CAs*	**Resultado**
Peq	273 50.000 + 273 8	13.332 KB
Méd	273 600.000 + 273 40	159.971 KB
Grd	273 3.000.000 + 273 100	799.831 KB

Esquema Proposto:

Neste esquema vamos ter um registro de TabFolha com 49 bytes, para cada cliente-folha, e teremos um registro de TabFilha com 113 bytes, para cada CA-Filha. Portanto:

PKI	49 *CliFol* 113 *CAs*	**Resultado**
Peq	49 50.000 + 113 8	2.393 KB
Méd	49 600.000 + 113 40	28.715 KB
Grd	49 3.000.000 + 113 100	143.565 KB

Representando nos três exemplos uma *economia de 82% em armazenamento nos OCSP-Rs*.

Por desempenhar um papel singular, o OCSP-Anc deverá armazenar a mais do que os OCSP-Rs normais, a TabPathOCSP (cujo registro é de 64 bytes e o número de registros é igual ao número de CAs descendentes) e a TabChaveOCSP (com registros também de 64 bytes e o número de linhas igual ao número de CAs-Filha do OCSP-Anc). Também deverá armazenar os dados enviados pelo verificador em sua consulta (97 *l* 15 bytes). Porém estes valores não são significativos (<0,001%) em relação aos dados já armazenados pelos OCSP-Rs normais.

7.3.3 Custo de Armazenamento no Cliente

Esquema Tradicional:

O cliente deve ter armazenado os *l* certificados do caminho, além das *l* respostas dos OCSP-Rs.

PKI	*l* 1.024 *l* 273	**Resultado**
Peq	4 1.024 + 4 273	5.188 Bytes
Méd	8 1.024 + 8 273	10.376 Bytes
Grd	15 1.024 + 15 273	19.455 Bytes

Esquema Proposto:

As variáveis armazenadas no cliente são: PathIDCli (32), o registro enviado pelo verificador (65) e os certificados pertencentes ao caminho, totalizando: 1.024 *l* 97 bytes.

PKI	*l*	**Resultado**
Peq	4	4.193 Bytes
Méd	8	8.289 Bytes
Grd	15	15.457 Bytes

Isto significa uma *economia de 20% em armazenamento no cliente*.

A tabela abaixo mostra um resumo das porcentagens das economias médias conseguidas com o método proposto em relação ao método tradicional do OCSP levando em consideração as PKIs sugeridas como exemplo.

Tabela 3 – Resultados obtidos com o novo esquema em relação ao esquema tradicional.

Área	Localização	Economia Média(%)
Processamento	Carga	99%
	OCSP-Anc	Custo *O(10)*
	Cliente	99%
Transmissão	Carga	82%
	Verif. OCSP-Rs	28%
	Verif. Cliente	96%
Armazenamento	Verif.	13%
	OCSP-Rs	82%
	Cliente	19%

8 CONCLUSÃO

Vimos que os custos extras envolvidos na implementação deste método de verificação não são grandes, ou seja, praticamente eliminando a carga que o sistema tradicional impõe no início de cada período de revogação às CAs e aos OCSP-Rs e direcionando o trabalho para o OCSP-Anc. Entretanto, este processamento do OCSP-Anc limita-se a cálculos de hashes, o que resulta um tempo de resposta da ordem de dezenas de microssegundos. O envio das provas de validade dos certificados pertencentes ao caminho de certificação fica resumido ao envio de apenas dois hashes (HDNfol e Carimbo), reduzindo em muito a transmissão e o processamento.

Esta abordagem permite um esforço muito menor dos clientes, que não precisam utilizar operações de chave pública para verificar se um certificado foi revogado.

9 REFERÊNCIAS BIBLIOGRÁFICAS

[1] M. Myers, et al, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP", RFC 2560, IETF PKIX Working Group, Jun/1999
Disponível em: http://www.ietf.org/rfc/rfc2560.txt, Últ. Acesso: Out/2003

[2] C. Adams, S. Lloyd, "Understanding Public-Key Infrastructure – Concepts, Standards, and Deployment Considerations", Livro, New Riders Publishing, Nov/1999

[3] S. Micali, "NOVOMODO Scalable Certificate Validation and Simplified PKI Management", Artigo, 1st Annual PKI Research Workshop, Ago/2002, Pág. 15 – 25
Disponível em: http://www.cs.dartmouth.edu/~pki02/Micali/paper.pdf, Últ. Acesso: Out/2003

[4] NIST, “Secure Hash Standard (SHS)”, Especificação, FIPS PUBS 180-2, Ago/2002
Disponível em: http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf, Últ. Acesso: Out/2003

[5] NIST, “Advanced Encryption Standard (AES)”, Especificação, FIPS PUBS 197, Nov/2001
Disponível em: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, Últ. Acesso: Out/2003

[6] W. Freeman, E. Miller, “An Experimental Analysis of Cryptographic Overhead in Performance-Critical Systems”, Artigo, MASCOTS’99, Mar/1999, Pág. 348 – 357
Disponível em: http://www.cse.ucsc.edu/~elm/Papers/mascots99.pdf, Últ. Acesso: Out/2003

[7] COSIC, “COSIC – COmputer Security and Industrial Cryptography Research 1998”, Relatório Anual, Katholieke Universiteit Leuven, 1998
Disponível em: http://www.esat.kuleuven.ac.be/cosic/research/1998/, Últ. Acesso: Out/2003

[8] J. Lloyd, “The Botan Project – Benchmarks”, Software, Botan WEB Page
Disponível em: http://opencl.randombit.net/bmarks.php, Últ. Acesso: Out/2003

[9] Baltimore Technologies Ltd., “KeyTools Pro v5.1 Developer’s Guide”, Manual, Baltimore Technologies Ltd., Mai/2001
Disponível em:
http://download.baltimore.com/keytools/docs/v51/pro/j-docs/html/devguide/projdevguide-C.3.html,
Últ. Acesso: Out/2003

[10] W. Dai, “Crypto++ 5.1 Benchmarks”, Relatório, Eskimo North Inc, Jul/2003
Disponível em: http://www.eskimo.com/~weidai/benchmarks.html, Últ. Acesso: Out/2003

[11] NESSIE Partners, “Performance of Optimized Implementations of the NESSIE Primitives”, Relatório NES/DOC/TEC/WP6/D21/2, NESSIE Project, Fev/2003
Disponível em:
http://citeseer.nj.nec.com/cache/papers/cs/27336/http:zSzzSzwww.cosic.esat.kuleuven.ac.bezSznessiezSzdeliverableszSzD21-v2.pdf/preneel03nessie.pdf,
Últ. Acesso: Out/2003