

# 암호프로그래밍

## CRYPTOGRAPHY PROGRAMMING

### 10. 인증서와 공개키기반구조

정보보호학과  
이병천 교수

# 차례

2

- 1. 강의 개요
- 2. 암호와 정보보호
- 3. 프로그래밍 환경 구축 - 웹, 파이썬
- 4. 해시함수
- 5. 메시지인증코드
- 6. 패스워드기반 키생성
- 7. 대칭키 암호
- 8. 공개키 암호
- 9. 전자서명
- **10. 인증서와 공개키기반구조(PKI)**

## 10. 인증서와 공개키기반구조

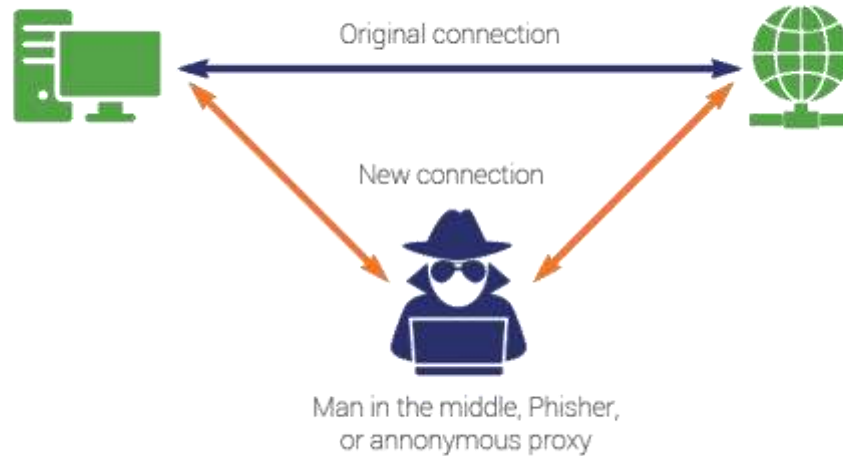
1. 인증서
2. OpenSSL
3. Forge 인증서 프로그래밍
4. 공개키기반구조(PKI)

Python 및 pycryptodome에는 인증서 관련 API 없음

# 1. 인증서

4

- 인터넷에서 상대방의 신분을 확인하지 않으면?



신분확인 필요

인증서는 신분확인+보안통신을  
제공하기 위한 핵심 수단

Avoiding **Man-in-the-Middle** Attacks



보안통신 필요

# 인증서

5

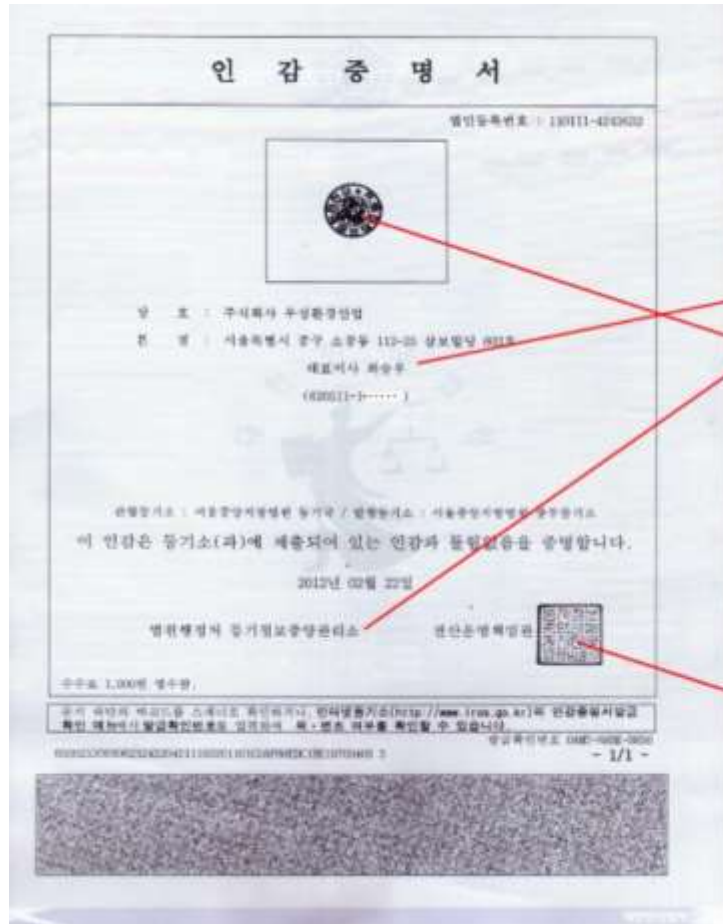
- 인터넷상의 신분증, 공인인증서
- 많은 타인들과 인증된 통신 가능



# 인증서

6

## □ 인감증명서 vs. 공인인증서



서명 전 인증서	
규격의 버전	3
인증서의 일련 번호	17:ab:4a:84:7d:6c:15:8e:79:4c:2e:e8:e8:26:7d:23:
디지털 서명의 알고리즘	md5WithRSAEncryption
인증서의 발행자	CrossCert Class 1 Consumer Individual Subscriber CA
유효 기한의 개시	Dec 24 00:00:00 2006 UTC
유효 기한의 종료	Feb 22 23:59:59 2007 UTC
공개 키의 소유자	GiDong Hong, /Email=gldong@novel.ac.kr
공개 키의 알고리즘	rsaEncryption
공개 키 본체	RSA Public Key: (1024 bit)  Modulus (1024 bit): e3:08:47:05:ea:69:6c:ef:d9:8c:59:a0:79:1c:4a:84:a5:44:91:3b: 92:4c:1c:09:4e:e6:c6:fb:88:67:42:3e:bb:1e:75:75:b9:38:97:35: dc:6b:20:ca:07:2d:71:fa:fa:d5:18:51:14:17:b5:a0:87:17:1e:08: 3a:cb:be:23:18:16:3d:a9:33:19:53:38:45:b7:e4:8a:31:65:5b:26: ac:d0:6a:48:c3:50:2d:b4:b2:bc:e0:16:1c:23:1d:39:8b:bd:93:0e: c1:ac:40:10:3f:e2:e8:4e:6e:20:88:6c:ab:24:b9:c5:5b:b1:fb:3f: 9a:10:46:0f:a1:57:9b:23:  Exponent: 00:01:00:01:
확장 항목(선택)	
디지털 서명 알고리즘	md5WithRSAEncryption
디지털 서명 본체	68:90:36:be:d8:16:c5:74:1c:52:c7:5e:b0:43:6e:03:25:9a: e6:5e:6c:cb:dc:c1:11:c0:2a:70:de:ba:12:28:80:fa:9b:fa: 20:7f:e7:47:16:11:21:a1:e6:d9:2a:3e:c4:8b:83:ce:d9:e4: 77:39:c1:61:0f:e5:4f:27:22:c1:ca:15:29:73:8d:10:58:48: 0e:75:28:0f:16:9e:10:76:ca:8d:8d:09:04:84:fd:a6:38:5e: a9:17:56:2d:fb:a8:23:dc:a4:45:58:bc:54:1b:17:67:c6:da: 8a:6b:ae:0e:71:db:7e:20:45:58:0c:67:97:de:00:8c:fb:51: e0:04:

## □ 인증서(Certificate)란?

- 개인이 사용하는 공개키를 인증기관이 인증해주는 전자문서
- 개인정보와 공개키가 포함된 전자문서를 인증기관이 전자서명한 문서
- 인증기관을 신뢰하는 범위 내에서 사용자 인증을 위해 널리 호환되어 사용 가능



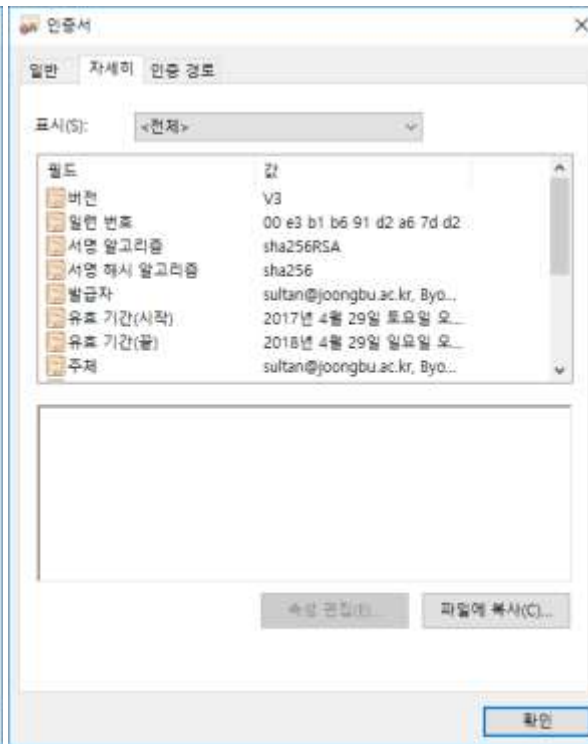
# 인증서 보기

8

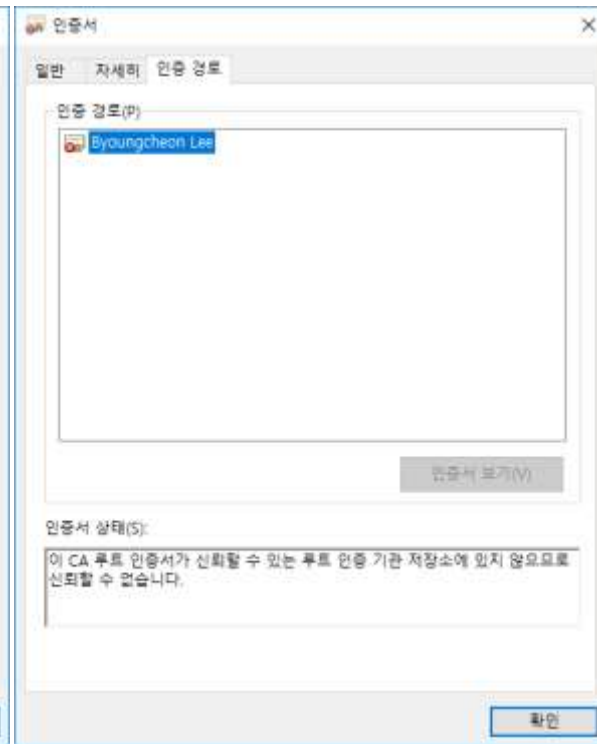
일반



자세히



인증경로





# 공동인증서

9

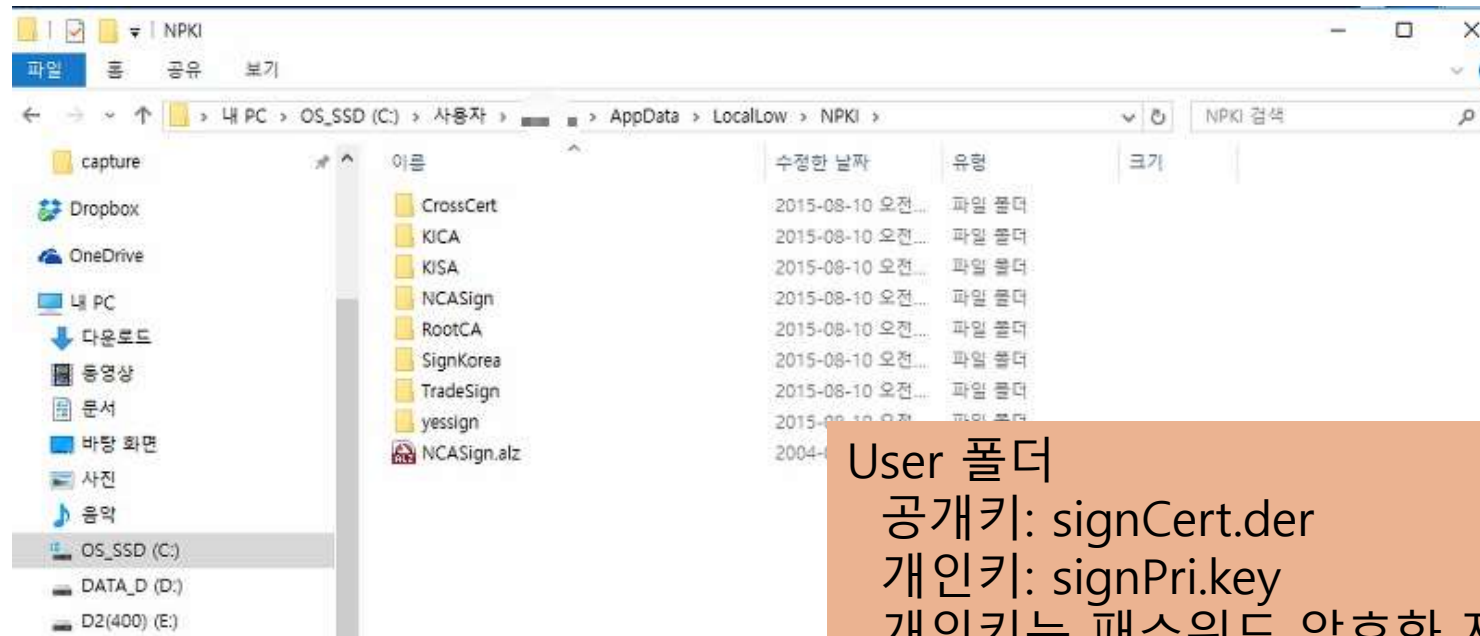
## □ 공동인증서의 저장 위치

### ▣ Windows XP

- C: > Program files > NPki

### ▣ Windows 7 이후

- C: > 사용자 > 사용중인 계정 이름 > AppData > LocalLow > NPki



User 폴더

공개키: signCert.der

개인키: signPri.key

개인키는 패스워드 암호화 저장됨

# 공동인증서 정보 보기

10

The screenshot shows a window titled '인증서' (Certificate) with three tabs: '일반' (General), '자세히' (Details), and '인증 경로' (Certification Path). The '자세히' tab is selected. It displays a table of certificate fields and their values.

필드	값
버전	V3
일련 번호	1e d1 d7 ce
서명 알고리즘	sha256RSA
서명 해시 알고리즘	sha256
발급자	yessignCA Class 2, AccreditedCA
유효 기간(시작)	2017년 3월 3일 금요일 오...
유효 기간(끝)	2018년 3월 18일 일요일 오...
주체	이병천(Byoungcheon Lee)O...

Below the table, the following information is displayed:

CN = yessignCA Class 2  
OU = AccreditedCA  
O = yessign  
C = kr

Buttons at the bottom: '속성 편집(E)...', '파일에 복사(C)...', and '확인'.

서명 해시 알고리즘: sha256  
공개키: RSA 2048 bits  
키 사용: 전자서명, 부인방지  
인증서 정책:

<http://www.yessign.or.kr/cps.htm>

The screenshot shows the '인증업무준칙(CPS)' (Certification Business Guidelines) page. It includes a header with the title and a sub-header 'yessign 인증업무준칙에 대해 안내해 드립니다.' (We will guide you about the yessign certification business guidelines).

The main content area contains a notice: '본 인증업무준칙은 금융결제원 전자인증센터가 발급하는 yessign 인증서의 발급, 이용 등에 관한 전반적인 사항 및 금융결제원 yessign 인증서비스 관련 당사자와 외부와 책임을 규정한 것으로 전자서명 관련 법률을 준수합니다.' (This certification business guideline is issued by the Korea Financial Infrastructure Security Center (KFIS) to guide the issuance and use of yessign certificates. It covers all matters related to the yessign certification service and the responsibilities of the company and external parties, and it complies with the laws related to electronic signatures.)

Below the notice, there is a '전문 다운로드' (Download) button. The page is divided into two columns of content:

- 01 개요** (Overview)
  - 1.1 비준 및 목적
  - 1.2 준칙의 명칭
  - 1.3 공인인증서발행인용처에 관한자
  - 1.4 준칙의 관리
  - 1.5 절차 및 처리
- 02 공인인증서 종류 및 수수료** (Types and Fees of Public Key Certificates)
  - 2.1 공인인증서 종류
  - 2.2 공인인증업무 수수료
- 04 공인인증업무 관련정보의 광고** (Advertising of Information Related to Public Key Certificate Business)
  - 4.1 광고 실비
  - 4.2 광고 방법
- 05 공인인증업무 시설 및 장비 보호조치** (Facility and Equipment Protection Measures for Public Key Certificate Business)
  - 5.1 물리적 보호조치
  - 5.2 절차적 보호조치
  - 5.3 기술적 보호조치
  - 5.4 인적 보안
  - 5.5 접근사 기록

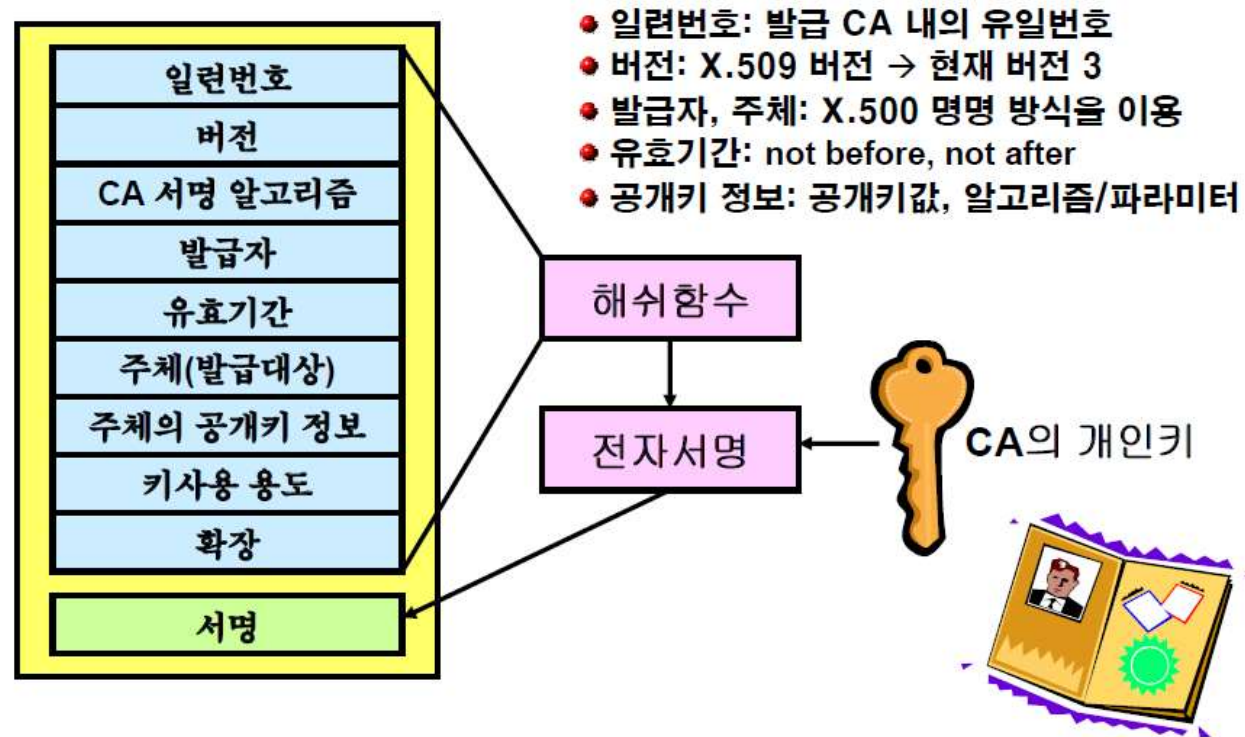
# X.509 인증서

11

## □ 인증서 표준

- X.509, The Directory: Authentication Framework, 1993.

## □ 인증서의 구성



## 2. OpenSSL

12

- Secure Sockets Layer (SSL) 이란?
  - ▣ 넷스케이프 사에서 최초 개발
  - ▣ 서버와 클라이언트 사이의 통신 보안을 제공하는 프로토콜
  - ▣ IETF 에서 TLS (Transport Layer Security) 라는 이름으로 표준화
- OpenSSL이란?
  - ▣ SSL/TLS 통신보안을 제공하는 오픈소스 툴킷
  - ▣ 인증서 생성 및 이용 기능 제공
  - ▣ <https://www.openssl.org/>

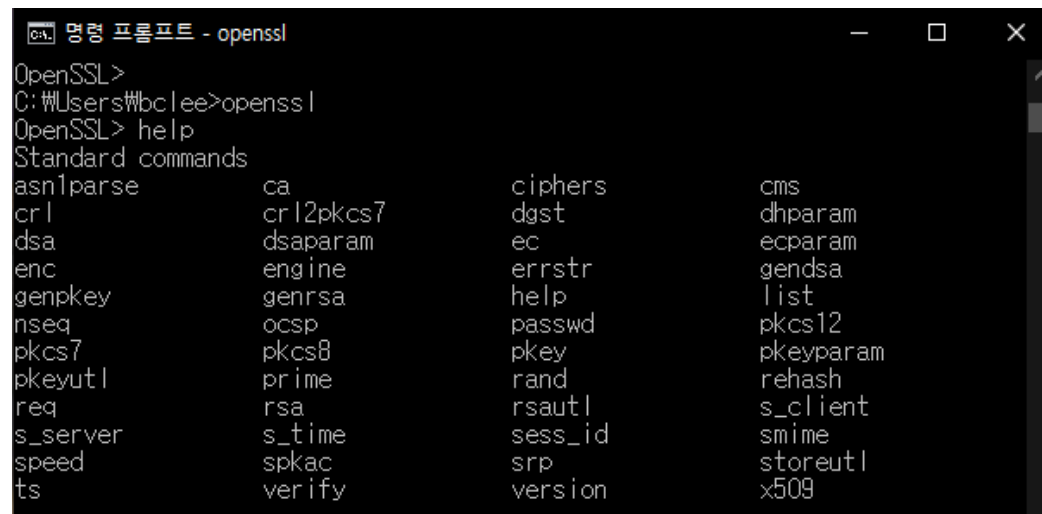


# OpenSSL 설치

13

- 리눅스 환경
  - ▣ 기본 설치됨
- 윈도우 환경
  - ▣ <http://slproweb.com/products/Win32OpenSSL.html>
  - ▣ Win64 OpenSSL v1.1.1k 다운로드 및 설치
  - ▣ 설치 디렉토리: C:\OpenSSL-Win64
  - ▣ 환경변수에 설치 디렉토리 등록, C:\OpenSSL-Win64\bin

설치 확인



```
명령 프롬프트 - openssl
OpenSSL>
C:\Users\Wbclee>openssl
OpenSSL> help
Standard commands
asn1parse      ca             ciphers        cms
crl            crl2pkcs7     dgst           dhparam
dsa            dsaparam      ec             ecparam
enc            engine        errstr         gendsa
genpkey        genrsa        help           list
nseq          ocsf          passwd         pkcs12
pkcs7          pkcs8         pkey           pkeyparam
pkeyutil       prime         rand           rehash
req            rsa           rsautl         s_client
s_server       s_time        sess_id        smime
speed          spkac         srp            storeutl
ts             verify        version        x509
```

# OpenSSL 사용법

14

## □ 1. RSA 키생성

> openssl genrsa -des3 -out private.pem 2048

```
F:\AppliedCrypto\mongochat>openssl genrsa -des3 -out private.pem 2048
Generating RSA private key, 2048 bit long modulus
.....+++
.....+++
unable to write 'random state'
e is 65537 (0x10001)
Enter pass phrase for private.pem:
Verifying - Enter pass phrase for private.pem:
```

개인키의 암호화 저장을 위한  
패스워드 입력 요구

PEM 인코딩된 텍스트 파일로 저장

```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4, ENCRYPTED
DEK-Info: DES-EDE3-CBC, 3686B2E9F3143376

3x814UV9n/ATI0S3dYm5Vne0ThOf1UYejdUdzYCTeq1wCnTZMkBNajhRKZ/OtVYX
i+eBjd/NmGTFRLSQmpFRst8VQ83WSTG+47q+tIerySxGj7WTuWiPvcgo/Hnr32A5
4AhVShVBSTiVyxSChyz/01YEurUB2y5LHS27H+gnWI/QubH6CTjIRfbX743vccQw
pH6cD1bw0w+oSseIVY7tKm/lvdiBqMcqr420aHlqsVcmJTSUK5fsY8VNfMZwtG7A
2dezoecxedvT/DvIoXGbaz0AIuJxB1+S12QzvTEpzOyNIi6OL+HjFNpOUumKQ77L
Wc2Fc4dz0gV81vxGBBjAfgi3kLkzqYp7hP3NLhOcfD2Fi/0q10iru06A6dywL8CM
12JyWSyi90W6PdoTeVJyHwqHm8ywUXhveeVyF7aoezbGhJu5jFc4lKY8qEpnCbd
M2vXFVDt8fKrH2o79TSxdJuZmR1AzTtJ3AzaZ76zJx3PldmnWjwJoRBikGnASdBf
PBF5pLJzFGKkobWT7dWzeQHE5ImMvru9CTdTF9gOAWepGEkr9IgbvHC/8XWp7Y8
3ASfEy2dfwNiug62rmWY12DRIwh0AbQhxxYYJmdlcZHqSpVcI0R0Yy33N41P1DFk
uDVNkcCVV3YDMAfNiyuCYanVbqFXysU/8Gjq6P9xtJwCx/ozBShjgriNxxgA0xJPr
t6xv6Aa5kMJ1PQQUxcUAmISjF1A710sihOfZ/ShQlriY+Ib0u4z5MymV1lqGaOMI
R8qrWSjzjNvkzBVm2VGJh9Zc+VS4S5n5Tv2GGYuXwHmzH9YMwx2UYyML/Ye5RsE7
fxXB862w9TK6j1fjGNqK1sHNxYHDyDY31SdiBRrz9EAPcchPH2/cRFOKGxdtLI+B
dMk7m8NIi9k00KSOqkOT7ZIT/RMcT9YeiOSMOMfsPqNy6CqccJHI5cAcdd8rXkuhO
sRW7udjnvQta57f1XQ1vvjQlwcWeKnNo0ioFWwBGiGUUx01PKMRbE3dSMmizPE8i
OxocLkbZilVggO1lJmZcCMojyCspkA44CM1nU6+XOgdHB45gMDFliGwkmmSLwMvQ
6jjIrerO23W18bVckmhqYhbjryFchlqoDdBcKF1VXYEYnwafrQ8XXCSo3XxVwlg
gDWPo5J2HSeCvfaLUFaqEmPvGr57cSEnudLkItw6Pk+9EIz8WF3iGELakbqMyvEJ
Jv3OZh6XxEcerdlD8glSCbJ50+1ESOS073AIfpgP/4RcCCfg6YZLIQvPgPeORaQb
wxkObMBGp7vd0WK++qJ/vHC/tvCB/UZGrdUdnLCLt6n2ZqPpvnI34y+gNmjTBxMi
+lwTH9bp9afv/3ebPSzqz5MkcmEQKpzsKyM7paRObLwAchLEuxhJ+BT/uMrtmJA
Lr5Ihlg0RGNH4qNutYnqjJVfEMZqn4ms/1XSfbcGHZSFttSZt+nRfiWSZjUgR98
PPGWCdzRuELOncnY0Oc5Xqfdl9RSJZ9nE9cYr1HK1bj7xUz5kM+v3OMjuQW5/ks0
ur2kkZi7LbvbFW69wuHrD8P+gi+qA4DEdJUIQ1YpYjKbmjvYJXSkzXSL8t0kDySy
-----END RSA PRIVATE KEY-----
```

# OpenSSL 사용법

15

- 2. 개인키로부터 공개키 추출
  - ▣ 개인키에는 공개키 정보가 내장되어 있음

```
> openssl rsa -in private.pem -outform PEM -pubout -out public.pem
```

```
F:\AppliedCrypto\mongochat>openssl rsa -in private.pem -outform PEM -pubout -out public.pem
Enter pass phrase for private.pem:
writing RSA key
```

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA4HIRqVC4Vonp694Veeyo
U9YfhbiyquJ6z71KVGdsk0AKOkAY/xnmjTEN4fY7f81X8x7ySjcWV0Q46SLnHQwN
l5ysBJLGL7mQIarHSHRv0vOoFUVWKTkXYZ4sdPpXKuNfvGrS0NkP2SBXywtcqF3f
rx1zJPPepJI3/drq2x7AnspMZCiXd7Sf2YO1YLrVMZxYlIkjqcDxiJy+kJiBmcAT
o4xkxsQK/kcRXMODTezA9g7npk0dO6KErx6OB/8UQkaMSdlqkPJxUnn2WcKbR4dq
lFqMLfg0bI7LoIDtBE+Lwams3H96+vJc0CJoECQTGXs2WdG1zMLSMrD+4jMwvOyk
3QIDAQAB
-----END PUBLIC KEY-----
```

## 16

■ 자신의 개인키로 자신의 공개키를 서명한 자체서명인증을 생성하여 사용

```
> openssl req -new -x509 -days 365 -key private.pem -out ca.crt
```

```
F:\AppliedCrypto\mongochat>openssl req -new -x509 -days 3650 -key private.pem -out ca.crt
Enter pass phrase for private.pem:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:KR
State or Province Name (full name) [Some-State]:Gyeonggi-do
Locality Name (eg, city) []:Goyang-si
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Joongbu u
Organizational Unit Name (eg, section) []:Dept. of Information Secur
Common Name (e.g. server FQDN or YOUR name) []:Byoungcheon Lee
Email Address []:sultan@joongbu.ac.kr
-----BEGIN CERTIFICATE-----
MIIESzCCAzOgAwIBAgIJAOmfnwNT
VQQGEWJLUjEUMBIGALUECAwLR3l1
aTEbMBkGALUECgwSSm9vbmdidSB1
ZiBJbmZvcmlhdGlvbiBTZWNNcm10
MSMwIQYJKoZIhvcNAQkBfHRzdWx0
NjEwMDhaFw0YnZExMDGwNjEwMDha
R3l1b25nZ2ktZG8xejAQBgNVBAAC
dSB1bml2ZXJzaXR5MSYwJAYDVQQL
cm10eTEYMBYGA1UEAwwPQnlvdW5n
dWx0YXNSAam9vbmdidSB5Y5rcjCC
ggEBAOBByEalQuFaJ6evefXnsqFPW
03/JV/Me8ko3F1dEOoki5x0MDdec
VuxiX7kgGcDZD8kV9eLYh3d368
```

[illegible]

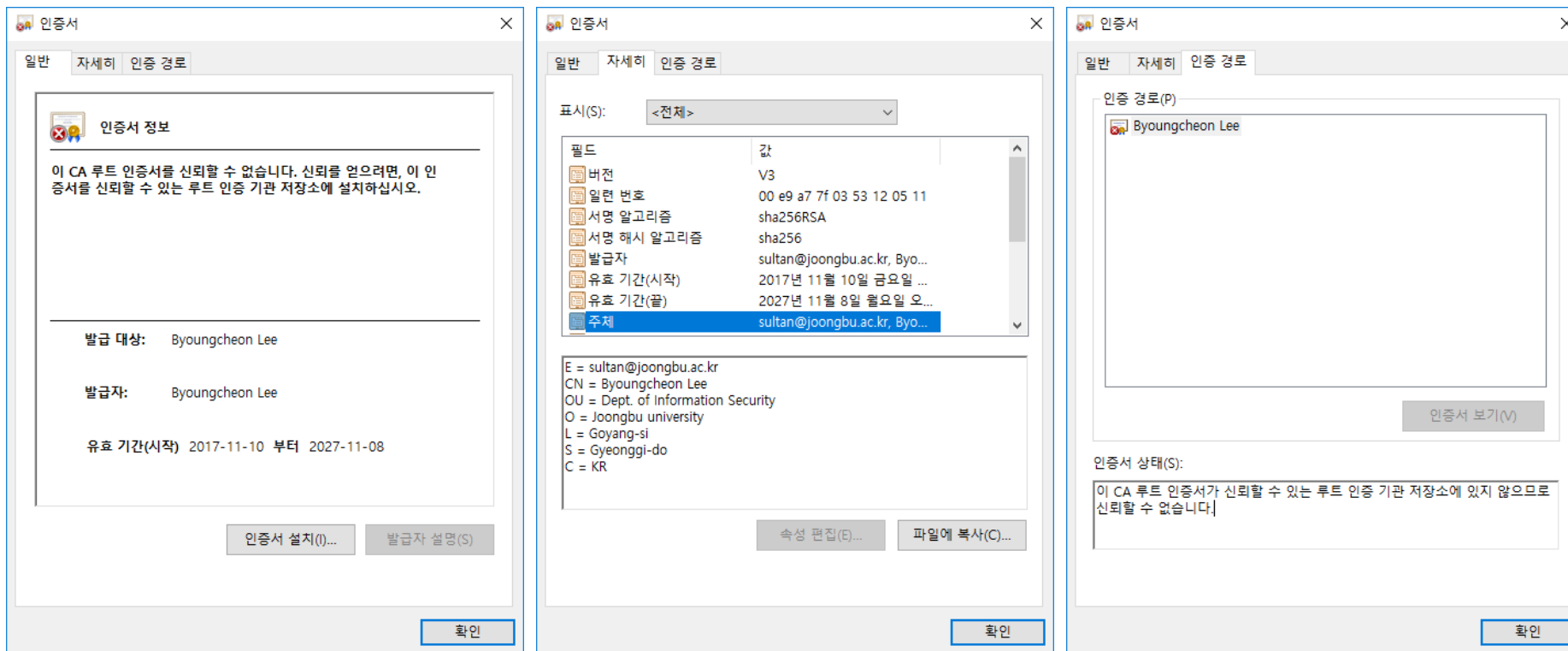
## 각종 사용자 정보 입력 요구



# OpenSSL로 자체서명인증서 생성

17

- 생성된 인증서 보기
  - ▣ ca.crt 파일 클릭



# 인코딩과 저장형식

18

- Base64
- ASN.1 (Abstract Syntax Notation One)
- DER (Distinguished Encoding Rules)
- PEM (Privacy-enhanced Electronic Mail)

# Base64 인코딩

19

## □ Base64 인코딩

- 8비트 이진 데이터(예를 들어 실행 파일이나, ZIP 파일 등)를 문자 코드에 영향을 받지 않는 공통 ASCII 영역의 문자들로만 이루어진 일련의 문자열로 바꾸는 인코딩 방식
- $64 = 2^6$ 은 화면에 표시되는 ASCII 문자들을 써서 표현할 수 있는 가장 큰 진법
- 알파벳 대소문자(A-Z, a-z)와 숫자(0-9) 그리고 두 개의 기호("+", "/")로 이루어져 있으며, 패딩 문자로 "="를 사용
- 바이너리 데이터 3바이트를 4개의 Base64 문자로 변환 ( $8 \times 3 = 6 \times 4$ )
- 입력되는 데이터가 3바이트로 나누어 떨어지지 않는 경우에는 "="를 패딩 문자로 사용
- 저장, 전송, 화면 표시 등에 편리. 그러나 데이터가 확대되는 단점

# ASCII 와 Base64

20

## ASCII (8비트)

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	#32;	Space	64	40	100	#64;	@	96	60	140	#96;	`
1	1	001	SOH (start of heading)	33	21	041	#33;	!	65	41	101	#65;	A	97	61	141	#97;	a
2	2	002	STX (start of text)	34	22	042	#34;	"	66	42	102	#66;	B	98	62	142	#98;	b
3	3	003	ETX (end of text)	35	23	043	#35;	#	67	43	103	#67;	C	99	63	143	#99;	c
4	4	004	END (end of transmission)	36	24	044	#36;	\$	68	44	104	#68;	D	100	64	144	#100;	d
5	5	005	ENQ (enquiry)	37	25	045	#37;	%	69	45	105	#69;	E	101	65	145	#101;	e
6	6	006	ACK (acknowledge)	38	26	046	#38;	&	70	46	106	#70;	F	102	66	146	#102;	f
7	7	007	BEL (bell)	39	27	047	#39;	'	71	47	107	#71;	G	103	67	147	#103;	g
8	8	010	BS (backspace)	40	28	050	#40;	(	72	48	110	#72;	H	104	68	150	#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	#41;	)	73	49	111	#73;	I	105	69	151	#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	#42;	*	74	4A	112	#74;	J	106	70	152	#106;	j
11	B	013	VT (vertical tab)	43	2B	053	#43;	+	75	4B	113	#75;	K	107	71	153	#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	#44;	,	76	4C	114	#76;	L	108	72	154	#108;	l
13	D	015	CR (carriage return)	45	2D	055	#45;	-	77	4D	115	#77;	M	109	73	155	#109;	m
14	E	016	SO (shift out)	46	2E	056	#46;	.	78	4E	116	#78;	N	110	74	156	#110;	n
15	F	017	SI (shift in)	47	2F	057	#47;	/	79	4F	117	#79;	O	111	75	157	#111;	o
16	10	020	DLE (data link escape)	48	30	060	#48;	0	80	50	120	#80;	P	112	76	158	#112;	p
17	11	021	DC1 (device control 1)	49	31	061	#49;	1	81	51	121	#81;	Q	113	77	159	#113;	q
18	12	022	DC2 (device control 2)	50	32	062	#50;	2	82	52	122	#82;	R	114	78	160	#114;	r
19	13	023	DC3 (device control 3)	51	33	063	#51;	3	83	53	123	#83;	S	115	79	161	#115;	s
20	14	024	DC4 (device control 4)	52	34	064	#52;	4	84	54	124	#84;	T	116	80	162	#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	#53;	5	85	55	125	#85;	U	117	81	163	#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	#54;	6	86	56	126	#86;	V	118	82	164	#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	#55;	7	87	57	127	#87;	W	119	83	165	#119;	w
24	18	030	CAN (cancel)	56	38	070	#56;	8	88	58	130	#88;	X	120	84	166	#120;	x
25	19	031	EM (end of medium)	57	39	071	#57;	9	89	59	131	#89;	Y	121	85	167	#121;	y
26	1A	032	SUB (substitute)	58	3A	072	#58;	:	90	5A	132	#90;	Z	122	86	168	#122;	z
27	1B	033	ESC (escape)	59	3B	073	#59;	;	91	5B	133	#91;	[	123	87	169	#123;	{
28	1C	034	FS (file separator)	60	3C	074	#60;	<	92	5C	134	#92;	\	124	88	170	#124;	
29	1D	035	GS (group separator)	61	3D	075	#61;	=	93	5D	135	#93;	]	125	89	171	#125;	}
30	1E	036	RS (record separator)	62	3E	076	#62;	>	94	5E	136	#94;	^	126	90	172	#126;	~
31	1F	037	US (unit separator)	63	3F	077	#63;	?	95	5F	137	#95;	_	127	91	173	#127;	DEL

Source: [www.LookupTables.com](http://www.LookupTables.com)

128	Ç	144	É	160	á	176	⌘	192	⌘	208	⌘	224	α	240	≡
129	ü	145	æ	161	í	177	⌘	193	⌘	209	⌘	225	β	241	±
130	é	146	Æ	162	ó	178	⌘	194	⌘	210	⌘	226	Γ	242	≥
131	â	147	ô	163	ú	179		195	⌘	211	⌘	227	π	243	≤
132	ä	148	ö	164	Ë	180	⌘	196	⌘	212	⌘	228	Σ	244	∫
133	à	149	ò	165	Ñ	181	⌘	197	⌘	213	⌘	229	σ	245	∫
134	â	150	û	166	*	182	⌘	198	⌘	214	⌘	230	μ	246	+
135	ç	151	ù	167	°	183	⌘	199	⌘	215	⌘	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	⌘	200	⌘	216	⌘	232	Φ	248	°
137	ë	153	Ö	169	⌘	185	⌘	201	⌘	217	⌘	233	Θ	249	.
138	è	154	Û	170	⌘	186	⌘	202	⌘	218	⌘	234	Ω	250	.
139	ï	155	°	171	½	187	⌘	203	⌘	219	■	235	δ	251	√
140	î	156	£	172	¾	188	⌘	204	⌘	220	■	236	∞	252	∞
141	ï	157	¥	173	⌘	189	⌘	205	=	221	■	237	φ	253	²
142	À	158	⌘	174	«	190	⌘	206	⌘	222	■	238	ε	254	■
143	Á	159	⌘	175	»	191	⌘	207	⌘	223	■	239	∩	255	

Source: [www.LookupTables.com](http://www.LookupTables.com)

## Base64 (6비트)

Value	Char	Value	Char	Value	Char	Value	Char
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

# Base64 인코딩

21

- ASCII 데이터 "Man"을 Base64 인코딩하면 "TWFu"로 변환

Text content	M								a								n							
ASCII	77								97								110							
Bit pattern	0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0
Index	19								22								5							
Base64-Encoded	T								W								F							

- 온라인 Base64 인코딩 테스트 사이트
  - ▣ <https://www.base64encode.org/>

## □ Abstract Syntax Notation One (ASN.1)

- ▣ 데이터 구조를 나타내기 위한 표현언어
- ▣ X.509 인증서를 저장하는데 사용
- ▣ 이기종 시스템간의 데이터 전송을 위한 표준 방식
- ▣ 적용 사례
  - PKCS group of cryptography standards
  - X.400 electronic mail
  - X.500 Lightweight Directory Access Protocol (LDAP)
  - H.323 (VoIP)
  - Kerberos
  - BACnet
  - simple network management protocol (SNMP)
  - UMTS, LTE, and WiMAX 2

# X.509 인증서의 ASN.1 표현

23

## □ X.509 인증서

```
Certificate ::= SEQUENCE {  
    tbsCertificate      TBSCertificate,  
    signatureAlgorithm  AlgorithmIdentifier,  
    signatureValue      BIT STRING }
```

아래 문서를 인증기관이 서명

```
TBSCertificate ::= SEQUENCE {  
    version          [0] EXPLICIT Version DEFAULT v1,  
    serialNumber      CertificateSerialNumber,  
    signature         AlgorithmIdentifier,  
    issuer            Name,  
    validity          Validity,  
    subject           Name,  
    subjectPublicKeyInfo SubjectPublicKeyInfo,  
    issuerUniqueID    [1] IMPLICIT UniqueIdentifier OPTIONAL,  
                    -- If present, version MUST be v2 or v3  
    subjectUniqueID   [2] IMPLICIT UniqueIdentifier OPTIONAL,  
                    -- If present, version MUST be v2 or v3  
    extensions        [3] EXPLICIT Extensions OPTIONAL  
                    -- If present, version MUST be v3  
}
```

인증서 양식

# 인증서 인코딩

24

## □ DER (Distinguished Encoding Rules)

- ▣ 바이너리 인코딩 (화면 표시가 어려움)
- ▣ 인증서 저장에 사용
- ▣ CER 또는 CRT 확장자로 사용되기도 함

```
openssl x509 -in certificate.der -inform der -text -noout
```

## □ PEM (Privacy-enhanced Electronic Mail)

- ▣ Base64 인코딩 (화면 출력 가능)
- ▣ X.509v3 에서 사용되는 여러 파일 양식들을 저장하는데 사용
  - 인증서, 개인키, 인증서 발급 요청 양식 등
- ▣ "— BEGIN ..." 으로 시작됨

```
openssl x509 -in cert.pem -text -noout  
openssl x509 -in cert.cer -text -noout  
openssl x509 -in cert.crt -text -noout
```



# 인증서 관련 확장자

25

- .DER
  - ▣ 인증서 저장에 사용되는 확장자
- .CRT
  - ▣ 인증서 저장에 사용 (Unix 방식)
  - ▣ The CRT extension is used for certificates. The certificates may be encoded as binary DER or as ASCII PEM. The CER and CRT extensions are nearly synonymous.
- .CER
  - ▣ 인증서 저장에 사용 (마이크로소프트 방식)
  - ▣ alternate form of .crt (Microsoft Convention)
- .KEY
  - ▣ PKCS#8 형식으로 공개키, 개인키 저장에 사용
  - ▣ The keys may be encoded as binary DER or as ASCII PEM

# 3. Forge 인증서 프로그래밍

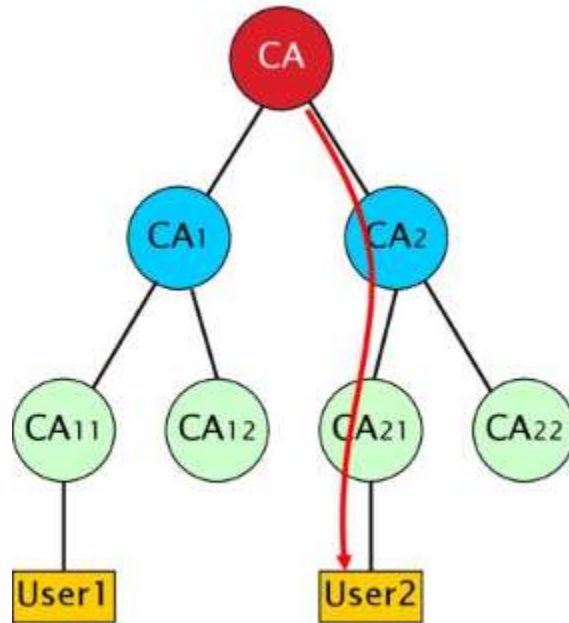
26

- 3.1 자체서명인증서 생성
  - ▣ 루트인증기관 준비
- 3.2 각종 부가 기능
  - ▣ 개인키, 인증서의 파일 저장
  - ▣ 개인키, 인증서를 파일에서 읽어오기
  - ▣ 개인키로부터 공개키 추출
  - ▣ 인증서에서 공개키 추출
  - ▣ 개인키의 패스워드 암호화
  - ▣ 패스워드 복호화하여 개인키 복구
- 3.3 인증기관에 의한 인증서 발급
  - ▣ 사용자 A에게 인증서 발급
  - ▣ 사용자 B에게 인증서 발급
- 3.4 인증서 활용
  - ▣ 전자서명 로그인, 서명된 메시지 전송, 전자봉투 전송

## 3.1 자체서명인증서 생성

27

- 루트인증기관은 최상위 인증기관
  - ▣ 자체서명인증서를 발급하여 사용
  - ▣ 자신의 개인키로 자신의 공개키에 대해 인증서 발급



# 자체서명인증서 생성

28

```
var forge = require('node-forge');  
var fs = require('fs');  
var pki = forge.pki;  
var rsa = forge.pki.rsa;
```

```
// generate a keypair and create an X.509v3 certificate  
var keypair = pki.rsa.generateKeyPair(1024);  
var publicKey = keypair.publicKey;  
var privateKey = keypair.privateKey;
```

```
console.log(pki.publicKeyToPem(publicKey));  
console.log(pki.privateKeyToPem(privateKey));
```

```
var cert = pki.createCertificate();
```

<< cert 객체에 필드 정보 입력 >>

```
cert.sign(privateKey);
```

전체 개요

키쌍 생성

인증서 객체 생성

인증서 객체에  
필드 정보 입력

인증서 객체를  
개인키로 서명

```

var forge = require('node-forge');
var fs = require('fs');
var pki = forge.pki;
var rsa = forge.pki.rsa;

// 루트인증기관에서 자체서명인증서 생성, 저장

// 1. RSA 키쌍 생성
var keypair = pki.rsa.generateKeyPair(1024);
var publicKey = keypair.publicKey;
var privateKey = keypair.privateKey;

console.log(pki.publicKeyToPem(publicKey));
console.log(pki.privateKeyToPem(privateKey));

// 2. X.509v3 인증서 객체 생성
var cert = pki.createCertificate();

// 3. 각종 필드 정보 입력
cert.publicKey = publicKey;
cert.serialNumber = '01'; // DB 등에 일련번호 관리 필요
cert.validity.notBefore = new Date(); // 발급시간, 현재시간
cert.validity.notAfter.setFullYear(cert.validity.notBefore.getFullYear() +
    1); // 유효기간 1년으로 설정
// 사용자 정보
var attrs = [{
    name: 'commonName', // 사용자명
    //shortName: 'CN',
    value: 'example.org'
}, {
    name: 'countryName', // 국가
    //shortName: 'C',
    value: 'KR'
}, {
    name: 'stateOrProvinceName', // 주, 지역
    //shortName: 'RT',
    value: 'Gyeonggi-do'
}, {
    name: 'localityName', // 도시명
    //shortName: 'L',
    value: 'Goyang-si'
}, {
    name: 'organizationName', // 기관명
    //shortName: 'O',
    value: 'Joongbu Univ.'
}, {
    name: 'organizationalUnitName', // 부서명
    //shortName: 'OU',
    value: 'Dept. of Information Security'
}];

```

```

cert.setSubject(attrs); // 인증서 사용자(주체)로 설정
cert.setIssuer(attrs); // 발급자로 설정 (자체서명인증서
    인 경우 동일)
// 확장영역 설정
cert.setExtensions([
    {
        name: 'basicConstraints',
        cA: true // 인증기관의 인증서임을 나타냄
    }, {
        name: 'keyUsage', // 키용도 설정
        keyCertSign: true,
        digitalSignature: true,
        nonRepudiation: true,
        keyEncipherment: true,
        dataEncipherment: true
    }, {
        name: 'extKeyUsage', // 확장 키용도 설정
        serverAuth: true,
        clientAuth: true,
        codeSigning: true,
        emailProtection: true,
        timeStamping: true
    }, {
        name: 'nsCertType', // 인증서 타입
        client: true,
        server: true,
        email: true,
        objsign: true,
        sslCA: true,
        emailCA: true,
        objCA: true
    }, {
        name: 'subjectAltName', // 주체 별도 정보
        altNames: [{
            type: 6, // URI
            value: 'http://example.org/webid#me'
        }, {
            type: 7, // IP
            ip: '127.0.0.1'
        }
    ],
    // 주체 키 식별자
    name: 'subjectKeyIdentifier'
});

```

## 자체서명인증서 생성 예제

```

// 4. 인증서 객체를 개인키로 서명
cert.sign(privateKey);

// 5. 인증서를 PEM 형식으로 출력
var pem = pki.certificateToPem(cert);
console.log(pem);

// 6. 인증서의 검증
var verified = cert.verify(cert); // 인증서에 있는 공
    개키로 검증
console.log('인증서 검증: '+verified);

// 7. 인증서, 개인키를 파일로 저장하기
fs.writeFile("rootPublicKey.pem", pki.publicKeyTo
    Pem(publicKey), function(err) {
    if(err) {
        return console.log(err);
    }
});

fs.writeFile("rootPrivateKey.pem", pki.privateKeyT
    oPem(privateKey), function(err) {
    if(err) {
        return console.log(err);
    }
});

fs.writeFile("rootCert.pem", pki.certificateToPem(c
    ert), function(err) {
    if(err) {
        return console.log(err);
    }
});

```

# 인증서 cert 생성 과정

30

필드정보입력

인증서 객체 생성

공개키 지정  
일련번호 지정  
유효기간-시작  
유효기간-끝  
인증서주체 지정  
발급자 지정  
확장영역

인증서 서명

인증서 검증

```
var cert = pki.createCertificate();

cert.publicKey = publicKey;
cert.serialNumber = '01';
cert.validity.notBefore = new Date();
cert.validity.notAfter.setFullYear(cert.validity.notBefore.getFullYear() + 1);
cert.setSubject(attrs);
cert.setIssuer(attrs);
cert.setExtensions( ..... );

cert.sign(privateKey);

var verified = cert.verify(cert);
```

인증서주체(Subject, 사용자)와 발급자(Issuer, 인증기관)가 동일하므로  
자체서명인증서(루트인증서)임

인증서주체(사용자)와 발급자(인증기관)가 다른 경우

- setIssuer에 발급자 정보를 입력
- 사용자 공개키를 cert.publicKey 에 지정
- 발급자의 개인키를 이용하여 인증서 서명 생성 cert.sign(privateKey);



# 인증서 cert 생성

31

## 주체 정보 attrs

```
var attrs = [{
  name: 'commonName', // 사용자명
  //shortName: 'CN',
  value: 'example.org'
}, {
  name: 'countryName', // 국가
  //shortName: 'C',
  value: 'KR'
}, {
  name: 'stateOrProvinceName', // 주, 지역
  //shortName: 'RT',
  value: 'Gyeonggi-do'
}, {
  name: 'localityName', // 도시명
  //shortName: 'L',
  value: 'Goyang-si'
}, {
  name: 'organizationName', // 기관명
  //shortName: 'O',
  value: 'Joongbu Univ.'
}, {
  name: 'organizationalUnitName', // 부서명
  //shortName: 'OU',
  value: 'Dept. of Information Security'
}];
```

## 확장영역

```
cert.setExtensions([{
  name: 'basicConstraints',
  cA: true
}, {
  name: 'keyUsage',
  keyCertSign: true,
  digitalSignature: true,
  nonRepudiation: true,
  keyEncipherment: true,
  dataEncipherment: true
}, {
  name: 'extKeyUsage',
  serverAuth: true,
  clientAuth: true,
  codeSigning: true,
  emailProtection: true,
  timeStamping: true
},
```

키 사용 용도

## 인증기관의 인증서임을 나타냄

```
{
  name: 'nsCertType',
  client: true,
  server: true,
  email: true,
  objsign: true,
  sslCA: true,
  emailCA: true,
  objCA: true
}, {
  name: 'subjectAltName',
  altNames: [{
    type: 6, // URI
    value: 'http://example.org/webid#me'
  }, {
    type: 7, // IP
    ip: '127.0.0.1'
  }
}, {
  name: 'subjectKeyIdentifier'
}]);
```

인증서 타입

주체 별도 정보

주체 키 식별자

# 주체 정보 지정 방식

32

```
var attrs = [{
  name: 'commonName', // 사용자명
  //shortName: 'CN',
  value: 'example.org'
}, {
  name: 'countryName', // 국가
  //shortName: 'C',
  value: 'KR'
}, {
  name: 'stateOrProvinceName', // 주, 지역
  //shortName: 'RT',
  value: 'Gyeonggi-do'
}, {
  name: 'localityName', // 도시명
  //shortName: 'L',
  value: 'Goyang-si'
}, {
  name: 'organizationName', // 기관명
  //shortName: 'O',
  value: 'Joongbu Univ.'
}, {
  name: 'organizationalUnitName', // 부서명
  //shortName: 'OU',
  value: 'Dept. of Information Security'
}];
```

name	shortName
commonName	CN
countryName	C
stateOrProvinceName	ST
localityName	L
organizationName	O
organizationalUnitName	OU



# 공개키인증서의 확장 필드

33

- 발급자키식별자(**Authority Key Identifier**): 이 인증서를 확인할 때 사용할 발급자의 공개키를 독특하게 식별하는 식별자
  - ▣ 자체 서명 인증서를 제외한 모든 인증서의 필수 요소
- 주체키식별자(**Subject Key Identifier**): 이 인증서에 포함된 공개키를 독특하게 식별하는 식별자
  - ▣ 인증기관 인증서의 경우에는 필수 요소
- 키용도(**Key Usage**): 이 인증서에 바인딩되어 있는 공개키의 사용용도를 한정하기 위해 사용
  - ▣ 전자서명, 부인방지, 키 암호화, 데이터 암호화, 키 동의 등
  - ▣ 인증기관의 경우 **keyCertSign**, **cRLSign**이 설정되어 있어야 함
- **CRL 분배점(CRL distribution point)**: 이 인증서의 폐지 여부를 확인하기 위한 인증서 폐지 목록이 있는 위치
- **Basic-Constraints**: 이 인증서가 인증기관의 인증서임을 나타내기 위해 사용됨

# PEM 형식으로 출력하기

34

```
// 공개키를 PEM 형식으로  
console.log(pki.publicKeyToPem(publicKey));
```

```
// 개인키를 PEM 형식으로  
console.log(pki.privateKeyToPem(privateKey));
```

```
// 개인키는 암호화하여 저장  
// 개인키는 외부로 드러나지 않도록 철저히 관리
```

```
// 인증서를 PEM 형식으로  
console.log(pki.certificateToPem(cert));
```

```
F:\AppliedCrypto\forge>node cert.js  
-----BEGIN PUBLIC KEY-----  
MIGfMA0GCsGQSBTb3DQEBAAQAA4GNADCBiQKBgQCCDFJU2jdy8kf8LREduHLP8oz5W  
8kG1h1REDMwFD061ZjDaVMTFyTm91eQ9ZSp6AH1q201ynVRoZWWW/CUHDxcZqfr/W  
0x8vm7hHVWkQ0jpsJz01DHfC0zzkUo9b3GxtVBbeGi/ATKZxwuiLakW0cwYRnQm0  
VKBsBmHer//ohHGLEQIDAQAB  
-----END PUBLIC KEY-----  
  
-----BEGIN RSA PRIVATE KEY-----  
MIICXQIBAAKBgQCCDFJU2jdy8kf8LREduHLP8oz5W8kG1h1REDMwFD061ZjDaVMTF  
yTm91eQ9ZSp6AH1q201ynVRoZWWW/CUHDxcZqfr/W0x8vm7hHVWkQ0jpsJz01DHfC  
0zzkUo9b3GxtVBbeGi/ATKZxwuiLakW0cwYRnQm0VKBsBmHer//ohHGLEQIDAQAB  
AoGAB/jug1zqegWFL349ofLZXLAIEIC3gWvEIEdu+Teq112kxwK03Rq6C1PCCAhp  
LXZtezGpwE5VXewc3hugBIPjx2wW12TA108DC3CxeNpUqDY7sy+lesdfMZP41Lh9  
EJeJw4CRptykSWeQRe7i/Uj69y9Tj++6sLLmjr4WBAQ2mIECQC9TYcV9F3SspcG  
95RQ2yISkMDSsQSVbQv3Yuqk8gwCh1zqD18AP9nV7jrfSWXBNFPyLpeBnr1LUZvu  
twjby6L1AkEAsUOWOQ/XUXNYzrq2Tfr01WKV4PmU9FcrhEve0xAiRn62N2KOBSp5  
4Mu4QXQC6UsdeNcoc4dC21Hr4bvUVzFb9QJBAB71TDeLHq21cDyaZjiZe0YK/MQL  
yS2mN2w8mmWdTIvUdy9r9QTH73tEsHrFK/xNMkzjk+OMBtwh4vK15T9xZkCQA5+  
t2NrBzFa3EUzUvR5C1F+2UCnR/3e7ukHhymPPS5EIEPpJDH281NmdS+s+9v1z3gI  
U+Kiid4DSelWQhWGIaMOCQCaJz878eC/BazYvVINJf5eS96/VOKWB+A00wbDWCV/  
U18YD8E8WYbysYU/+GZ+09V04hGRyC3SD3Hres4DKyZS  
-----END RSA PRIVATE KEY-----  
  
-----BEGIN CERTIFICATE-----  
MIIDBCCAm2gAwIBAgIBATANBgkqhkiG9w0BAQUFADBpMQowEgYDVQQDEwt1eGft  
cGx1Lm9yZzELMAkGA1UEBhMCVVMxETAPBgNVBAGTCFZpcmdpbm1hMRMwEQYDYQOH  
EwpCbGFja3NiZXNMQ0wCwYDYDQKEwRUZXNOMQ0wCwYDYDQOLwRUZXNOMB4XDTE3  
MTEwOTA4NTQyM1oXDTE4MTEwOTA4NTQyM1owAQUjBgNVBAMTGA1UEAaMLXzhhbXBsZS5v  
cmcxZCZAJBgNVBAYTA1VTMRQwDwYDVQQIEwVhWwXJnaW5pYTETMBEGA1UEBxMK0mxh  
Y2tzYnVzZm9uZGA1UEChMEVGZzZDENMAAsGA1UECxEVGVzZdDENBnzANBgkqhkiG  
9w0BAQUFAAOBjQAwYkCgYEAxSVNo3cvJH/CORHbhy6eqM+VvJBpYZURAZMBQzu  
iGYw21TEcxk5vdXkPWUqegByKtjiMp1UaGV1vw1Bw13Gan6/1tmFL5u4R1cCkDo6  
bCc9CAx3wtM85FKPW9x17VQW3hovvEymccLoi2pftHMGZ0Jj1SgbAZh3q/61Rx  
i3kCAwEAAaOBuzCBuDAMBGNVHRMEBTADAQH/MASGA1UdDwQEAwIC9DA7B9NVHSUE  
NDAYBggrBgEFBQcDAQYIKwYBBQUHAWIGCCsGAQUFBwMDBggrBgEFBQcDBAYIKwYB  
BQUHAWgweQYJYIZIAAyb4QgEBBAQDAgD3MCwGA1UdEQQIMCOGG2h0dHABLy91eGft  
cGx1Lm9yZy93ZWJpZCZ0NTZyCfWAAATAdBgNVHQ4EFgQUQdKojAJpB1wd50904drU  
CqIVuDsWdQYJKoZIhvcNAQEFBQADgYEASOD8411mwXo0qM4AEyB+frq9jJ91sh3  
u/ic6r/5fPIpc+4Jo22rBERr11V7hzwJt8N1vWEGTqWcRn1pbPrpwRaDX0mo6Xz3  
mtntRCgA4z+heabHZbF93Coqfv00Pq3xxx/kcSKYx1i0Hk7oCWDcp9KBpxE61cRp  
tKGXPLLUcQQ=  
-----END CERTIFICATE-----
```

# 인증서 유효성 검증하기

35

- 인증서에 저장된 공개키로 인증서를 검증함
- 자체서명 인증서의 경우

```
caCert.sign(caPrivateKey);  
  
var verified = caCert.verify(caCert);  
console.log('인증서 검증: '+verified);
```

- 인증기관이 사용자 인증서를 발급한 경우
  - ▣ 인증기관 인증서로 사용자 인증서를 검증함

```
cert.sign(caPrivateKey);  
  
var verified = caCert.verify(cert);  
console.log('인증서 검증: '+verified);
```

caCert: 인증기관 인증서  
cert: 사용자 인증서

# 인증서/개인키를 파일로 저장하기

36

```
var forge = require('node-forge');  
var fs = require('fs');  
var pki = forge.pki;  
var rsa = forge.pki.rsa;  
  
// generate a keypair and create an X.509v3 certificate  
var keypair = pki.rsa.generateKeyPair(1024);  
var publicKey = keypair.publicKey;  
var privateKey = keypair.privateKey;  
  
fs.writeFile("publicKey.pem", pki.publicKeyToPem(publicKey), function(err) {  
  if(err) {  
    return console.log(err);  
  });  
  
fs.writeFile("privateKey.pem", pki.privateKeyToPem(privateKey), function(err) {  
  if(err) {  
    return console.log(err);  
  });  
  
fs.writeFile("cert.pem", pki.certificateToPem(cert), function(err) {  
  if(err) {  
    return console.log(err);  
  });  
  
fs.writeFileSync("cert1.pem", pki.certificateToPem(cert));  
  
fs.writeFileSync("privateKey1.pem", pki.privateKeyToPem(privateKey));
```

fs 객체 추가 – 파일처리기능  
fs는 node.js의 내장객체

공개키 파일 저장 (비동기식)

개인키 파일 저장 (비동기식)

인증서 파일 저장 (비동기식)

동기식 개인키 파일 저장  
동기식 인증서 파일 저장

## 3.2 각종 부가기능 활용

37

- 인증서/개인키를 파일에서 읽어오기
- 인증서에서 공개키 추출하기
- 개인키에서 공개키 추출하기
- 개인키의 패스워드 암호화
- 패스워드 복호화하여 개인키 복구

비동기식 읽어오기

```
var certPem = fs.readFileSync('cert.pem', 'utf8');
var privateKeyPem = fs.readFileSync('privateKey.pem', 'utf8');
var cert = pki.certificateFromPem(certPem);
var privateKey = pki.privateKeyFromPem(privateKeyPem);
```

```

-----BEGIN CERTIFICATE-----
MIIDBCCAm2gAwIBAgIBATANBgkqhkiG9w0BAQUFADBpMRQwEgYDVQQDEwt1eGft
cGx1Lm9yZzELMAkGA1UEBhMCVVMxETAPBgNVBAGTCFZpcmdpbm1hMRMwEQYDVQQH
EwpCbGFiZjA3NiRlLnM0OQwCwYDVQQKEwRlUzXNOMQ0wCwYDVQQLEwRlUzXNOMB4XDTE3
MTEwOTA5MTM1NVV0XDTE4MTMwOTA5MTM1NVV0aTEUaXNMLXZhb3B5B3ZS5v
cmcxZzA3BgNVBAYTA1VTMREwDwYDVQQLEwhtWwXJnaW5pYyTETMBEGA1UEBjMKQmxh
Y2t4YnV5ZyZlZEMMAA1UEChMEVGVYzdDENMAA1UECzMEVGVYzdDDBnZANBgkqhkiG
9w0BAQEFAA0BjQAwgYkCgYEAySGNSCgPqk+DjpDSBVW1/DDGEvQPvgS996f4NDXA
USWf407v1HKeDgyOpmCe7oTTcWKKCAQowRhZrR8XaMTbnpg7RM1O3PaxzsrfnXQf
o9Np7YAhD+U304F+jGcM+qgn78h37NCKq4SJUTs1EsbU5nj66Gs118Xd0cmcNVN9
8CAwEAaA0BuzCBuDAMBgNVHRMERTADAQH/MAsGAT1udDwQEAw1C9DA7BgNVHSUE
NDYABggrBgEFBQcDAQYIKwYBBQUHAW1G0CsGAQUFBwMDBggrBgEFBQcDBAYIKwYB
BQUHAgwEQGYJY1Z1AYb4QgEBBAQDAgD3MCwGA1UdEQQ1MCOGQ2h0dHA6Ly91eGft
cGx1Lm9yZy93ZWJpZCZ0ntZyCfWAAATAdBgNVHQ4EFgQU80/S/6+XWw/oz1IBv3gx
/haOH3wwDQYJKoZIhvcNAQEFBQADgYEACmhbEnY+JRswan2a/Y+pqD9M21jYwMVR
pWZJ/scB/Mo5r9vPohL0h0d2E9LMut1UqtjZ5P11h59dnMkMeZ44UAzTNyAOAJBa
nDwS1zZw9PPK3BJE3B0y/Bvzs2eBhtDG44t81lVQeBPHDs0sX6GaapUCbc55rNMP
oj8yppD0zBw=
-----END CERTIFICATE-----

-----BEGIN PUBLIC KEY-----
MIGfMA0GCsGgSIb3DQEBAGUAA4GNADCBiQKBgQDJY1Y1KA+qT400kN1FVbX8MMYS
9A/KBL30R/g0NcAhZRJZ/jTu/Ucp40DL5mYJ7uhNNxYo1BCjBEfNHxdoxNuemDTe
yU7c9rH0yt83FB/zij02ntgCEP5Tc74WMZwz6qqfvyHfsOKSrh11R0yUSxtTmePr
oazXXxd0KZw1U322HwIDAQAB
-----END PUBLIC KEY-----

```

# 동기식 읽어오기

# 인증서에서 공개키 추출하기

39

- 사용자의 공개키는 인증서의 필드로 등록되어 있으므로 인증서에서 다음과 같이 간단히 읽어올 수 있음

```
// 2. 인증서에서 공개키 추출하기
var publicKey = cert.publicKey;

console.log('Extract public key from Certificate: ');
console.log(pki.publicKeyToPem(publicKey));
```

# 개인키에서 공개키 추출하기

40

- 개인키 내부에 공개키 정보를 포함함
  - ▣ 개인키에서 공개키 추출 가능

```
// 3. 개인키에서 공개키 추출하기
var publicKey2 = pki.setRsaPublicKey(privateKey.n, privateKey.e);
console.log('개인키에서 추출한 공개키: ');
console.log(pki.publicKeyToPem(publicKey2));
```



# 개인키의 패스워드 암호화 저장

41

## □ PKCS#8

### ▣ 개인키를 패스워드로 암호화하여 파일로 저장

→ -----BEGIN PRIVATE KEY-----  
MIIBYgIBADANBgkqhkiG9w0BAQEFAASCauAwggE8AgEAAkEAq7BFUpkGp3+LQmIQ  
Yx2eqzDV+xeG8kx/sQFY18S5JhzGeIJNA72wSeukEPojtqUyX2J0CcIPBh7eqcIQ  
2zpAswIDAQABAgIsq4+zRdrzkWH1ITV1vpytnk0/NiHcnePQiOW0VUybPyHoGM  
/jf75C5xET7ZQpBe5kx5VHsPZjOCBb3b+wSRAIEA2mPWCBytosIU/ODRfq6EiY04  
lt6waE7I2uSPqIC20LcCICQDJQYIHQII+3YaPqyhGgqMexuuuGx+IDKD6/Fu/JwPb  
5QlhAKthiYcYKIL9h8bjDsQhZDUACPasjzdsDEdq8inDyLOFAIEAmCr/tZwa3qeA  
ZoBzI10DGP1uoKXBd3nk/eBxPkaxIEECIQCNymjsol7GI dtujYnr1qT+3yedLFHK  
srDYjIT3LsvTqw==  
-----END PRIVATE KEY-----

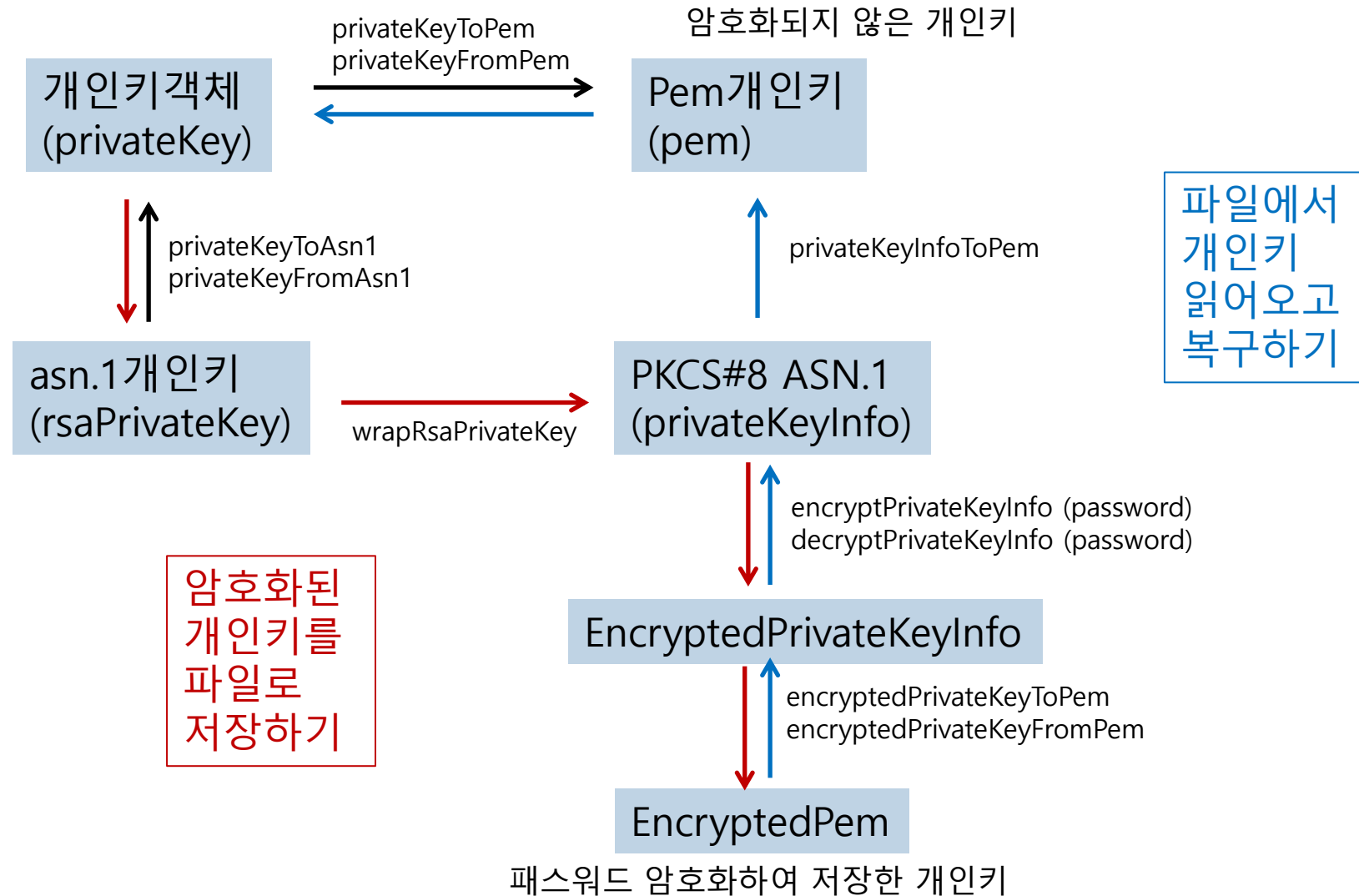
암호화되지 않은 개인키

→ -----BEGIN ENCRYPTED PRIVATE KEY-----  
MIIBrzBjBgkqhkiG9w0BBQowPDAbBgkqhkiG9w0BBQwwDgQImQ08S8BJYNACAggA  
MB0GCWCGSAFI AwQBKqQQ398SY1Y6moXTJC00PSahKgSCAWDeobyqIkAb9XmxjMmi  
hABtIIJBsybBymdlirtPjtRBTmz+ga40KFNFtKgTrtH0/3qf0wSHpWmKIQotRh6Ufk  
0VBh4QjbcNFQLzqJqbIW4E3v853PK1G40pQNpFLDLAPZLIyzxW0om9c9GXNm+ddG  
LbdeQRsPooIIdL61IYB505K/SXJCpemb1RCH0/dzsp/kRyLMQNSWiaJABkSyskcr  
eDJBZWOGQ/WJKI1CMHC8XgjqvmpXXas47G5sMSgFs+NUqVSkMSrsWMA+XkH/oT/x  
P8ze1vORDuQA1qaxdZh389h09BKfVcAFnLKK0tadIRkZHtNahVWnFUks5EP3C1k  
2cQQtWBkaZnRrEkB3H0/ty++WB0owHe7Pd9GKSntMIo8gmQzT2dfZP3+fIUFHTBs  
RZ9L8UWp2zt5hNDtc82hyNs70SETaSSaiygYNbBGIVAWVR9Mp8SMNYr1kdeGRgc3  
7r5E  
-----END ENCRYPTED PRIVATE KEY-----

암호화된 개인키

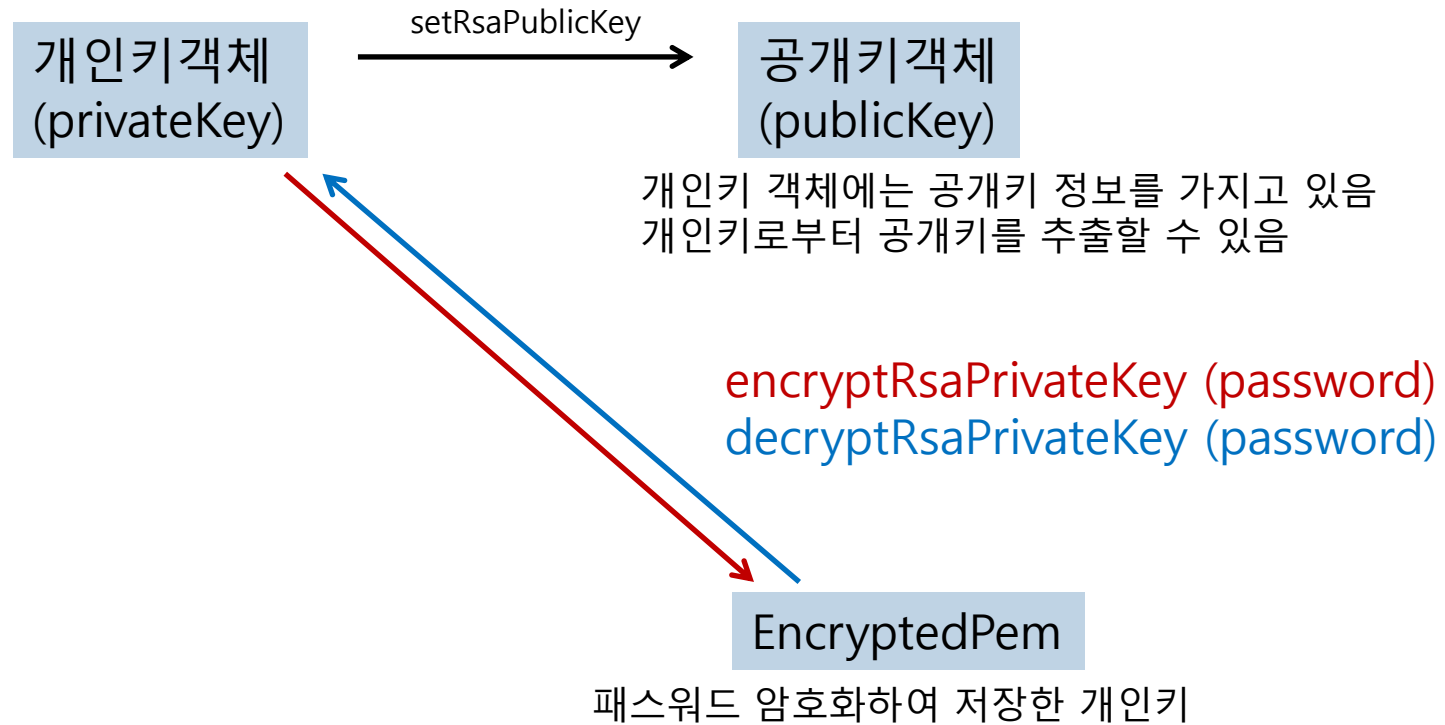
# PKCS#8 API

42



# PKCS#8 API

43



# PKCS#8 test – full API

```

var forge = require('node-forge');
var plaintext = "Hello world hello world";
var pki = forge.pki;
var rsa = forge.pki.rsa;

var keypair = rsa.generateKeyPair({bits: 1024, e: 0x10001});
var publicKey = keypair.publicKey;
var privateKey = keypair.privateKey;
var pubPem = forge.pki.publicKeyToPem(publicKey);
var privPem = forge.pki.privateKeyToPem(privateKey);

console.log('Public key: \n'+pubPem);
console.log('Private key: \n'+privPem);
console.log();

// PEM에서 개인키 읽어오기
var privateKey1 = pki.privateKeyFromPem(privPem);
console.log('Pem to Private key: \n'+pki.privateKeyToPem(privateKey1));

// 개인키를 ASN.1으로 출력
var privAsn1 = pki.privateKeyToAsn1(privateKey);
// convert an ASN.1 PrivateKeyInfo or RSAPrivateKey to a Forge PrivateKeyInfo
var privateKey2 = pki.privateKeyFromAsn1(privAsn1);
console.log('ASN.1 to Private key: \n'+pki.privateKeyToPem(privateKey2));

// ASN.1 개인키를 PrivateKeyInfo로 wrapping한 후에 pem으로 출력
// wrap an RSAPrivateKey ASN.1 object in a PKCS#8 ASN.1 PrivateKeyInfo
var privateKeyInfo = pki.wrapRsaPrivateKey(privAsn1);
// convert a PKCS#8 ASN.1 PrivateKeyInfo to PEM
var pem = pki.privateKeyInfoToPem(privateKeyInfo);
console.log('Private key Info: \n'+pem);

// PrivateKeyInfo를 패스워드 암호화/복호화
// encrypts a PrivateKeyInfo and outputs an EncryptedPrivateKeyInfo
var encryptedPrivateKeyInfo = pki.encryptPrivateKeyInfo(
  privateKeyInfo, 'password', {
    algorithm: 'aes256', // 'aes128', 'aes192', 'aes256', '3des'
  });

```

```

// decrypts an ASN.1 EncryptedPrivateKeyInfo
var privateKeyInfo1 = pki.decryptPrivateKeyInfo(
  encryptedPrivateKeyInfo, 'password');
console.log('Private key Info (enc/dec): \n'+pki.privateKeyInfoToPem(privateKeyInfo1));

// converts an EncryptedPrivateKeyInfo to PEM
var pem = pki.encryptedPrivateKeyToPem(encryptedPrivateKeyInfo);
console.log('EncryptedPrivateKeyInfo 1: \n'+pem);

// converts a PEM-encoded EncryptedPrivateKeyInfo to ASN.1 format
var encryptedPrivateKeyInfo = pki.encryptedPrivateKeyFromPem(pem);

// wraps and encrypts a Forge private key and outputs it in PEM format
var pem = pki.encryptRsaPrivateKey(privateKey, 'password');
console.log('EncryptedPrivateKeyInfo 2: \n'+pem);

// decrypts a PEM-formatted, encrypted private key
var privateKey2 = pki.decryptRsaPrivateKey(pem, 'password');
console.log('Pem to Private key 2: \n'+pki.privateKeyToPem(privateKey2));

// sets an RSA public key from a private key
var publicKey1 = pki.setRsaPublicKey(privateKey.n, privateKey.e);
console.log('Public key from Private key: \n'+pki.publicKeyToPem(publicKey1));

```

# PKCS#8 test – short API

45

```
var forge = require('node-forge');
var plaintext = "Hello world hello world";
var pki = forge.pki;
var rsa = forge.pki.rsa;
```

```
var keypair = rsa.generateKeyPair({bits: 1024, e: 0x10001});
var publicKey = keypair.publicKey;
var privateKey = keypair.privateKey;
```

```
console.log('Public key: \n'+forge.pki.publicKeyToPem(publicKey));
console.log('Private key: \n'+forge.pki.privateKeyToPem(privateKey));
```

```
// wraps and encrypts a Forge private key and outputs it in PEM format
var pem = pki.encryptRsaPrivateKey(privateKey, 'password');
console.log('EncryptedPrivateKeyInfo(password): \n'+pem);
```

```
// decrypts a PEM-formatted, encrypted private key
var privateKey1 = pki.decryptRsaPrivateKey(pem, 'password');
console.log('Decrypted Private key (password): \n'+pki.privateKeyToPem(privateKey1));
```

```
// sets an RSA public key from a private key
var publicKey1 = pki.setRsaPublicKey(privateKey.n, privateKey.e);
console.log('Public key from Private key: \n'+pki.publicKeyToPem(publicKey1));
```

```
F:\AppliedCrypto>node pkcs8-s.js
Public key:
-----BEGIN PUBLIC KEY-----
MIGfMA0GCsGSIb3DQEBAQUAA4GNADCBiQKBgQCFjGbu0B7qkbq33uksrwaHs0Wh
IAfzRTCj6LbOOVW8SRzkvj83Xaeu9oSSYolrS80z4GRDjzetzrhAA2qM8N53wCz
ZyqiW9/51zdo1BInQ/q5RGA83QPcENKZy5TWV6MiFobD9QwdGqTqPPviy5z1aY
Y9Gi6f5PDfoc1b8rQIDAQAB
-----END PUBLIC KEY-----
```

공개키

```
Private key:
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAKKBgQCFjGbu0B7qkbq33uksrwaHs0WhIAfzRTCj6LbOOVW8SRzkvj8
3Xaeu9oSSYolrS80z4GRDjzetzrhAA2qM8N53wCzZyqiW9/51zdo1BInQ/q5RGA
83QPcENKZy5TWV6MiFobD9QwdGqTqPPviy5z1aYIY9Gi6f5PDfoc1b8rQIDAQAB
AoGAPyYqvVksUj9Reh8jDukdoLrRltQxiqWfE70lQpUxkeuVCjibUJoQX/x86WT
h050yBb/tjb8qpKmwP8PlomlGnbkWF0qis0CBvmno+9hd0MVMv7f1EZb34UdNd
MLXGggoIY16K0YWAMfdWaoPtLA582aTOIA3PHURfLKi18YECQC75xvcmMk5nF2Y
mSMWn8zGa7YRXhWDNzM9qsikV503fngq/CmQ4dKOq2kslTs7CiAWJ2j9ppJqH
M1IRhr9AkeAu0/XJXYWBO+y7Hwgv69b3ml8N+OiklGtSLlT/gqzfxTIUzzukf
iygp/vebN2A8+00JAdhyX60Nuc0YiOBPCQAZpaq1g6P2pFhlH//ZUnH3oIONTs9
2Y41npSQyRfAt9t/qEpDN8NoWQGiBmoi7dud3T0EINW0dHZrsz6QLGX7QJAJeX
pNQ/biuk4NN5NevY4ZNA91uNs4+vThvugSx0Z78YCrSsYlTDxOVmc4hZUUISRj
mvPLSjbrI9W77/NIQJBAJ1x7rowwKdbSQ9NKO32sXInprZIKU57ws6mVxkFAQC
r/b0rEP9Z1eYS0i659ufjsPxTk+9zZAd0K5YKe9hTtl=
-----END RSA PRIVATE KEY-----
```

개인키

```
EncryptedPrivateKeyInfo(password):
-----BEGIN ENCRYPTED PRIVATE KEY-----
MIICzzBjBgkqhkiG9w0BBQowPDAwBgkqhkiG9w0BBQwwDgQIHmL3yHFJ0HkCaggA
MB0GCWCSGSAfIAwQBAGQ5H2i8FhSjcnJqsokJfXAHQSCAoDwthGJLfQI95bOxLU7
kW2KBMOTDETgmweLwxTO9/Ono0yrw9RzLMEoEO9BhlChvMI5ya2Ap9iUQBS9tvt
oRhHfSHqch+vQXdVc79ebpETrj32kNKNvB5WlnRdcUmefK4l/M+oIBxlHQX+QXD
mAHdALg2mx077+aCb49q0alpjDpXOLM/mFktrFm15dGNeYUNTOx65Zaynixf7rJs
4lb0mXr+yujr9dANOG6+O3fsJWHRL5VUKJ32RuXknRrig0i1BPRA7HUvNszUQXzj
rZL68MqnTOnqFTXnuWov93TrBduZUoPS00D2/qQhRym9FedVV48s9dHmQ4+d7Bm2
F4qRak/fuRmZOliY3ebKfTfKUKx+9fvAS6LUqEWK0ftdmuMCnwJUceWaz4RjWUE
ahSkas7Ud/bH9jesvu8BEU17juwf7v+m++Bbjw8G0740K2rdBzH3W0ssjld4C1P1
g5mXnbmdNs6661FW2F79dG/tZUv2bvEGZzzpVQRULHA8Hf1NOHPX05NsF/v6IX
XUjlyXRl8xd151g+pVvrujLvYBLUt4Tm3a7YqfJ4NM6dxrr/rBPP1SMdAyddA
mnmsvSjUeNhKz2zKDoobOKWKBMAIf5nXOMFE0+g5ciyfhUvgKGLCGPHKMNlm+fh0F
jMUCUED3UfrVQKNhMKdkvLjkl2NklFP2Gjv51s+YkH1ksXmmDq+le87vBKU9Bhd
93Kpk+kAsGBkDK07DDzTYvRAJ1BNI2tG2y/Oaj5bWkElIX4QOwNKzQZunvxXtdn
E2jPVAGESz3woDrgWSKF/TRGknRO1s2fslUx/qeo5Rs8UsuQSUslxbMK2aFlK
7cOM
-----END ENCRYPTED PRIVATE KEY-----
```

암호화된 개인키

```
Decrypted Private key (password):
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAKKBgQCFjGbu0B7qkbq33uksrwaHs0WhIAfzRTCj6LbOOVW8SRzkvj8
3Xaeu9oSSYolrS80z4GRDjzetzrhAA2qM8N53wCzZyqiW9/51zdo1BInQ/q5RGA
83QPcENKZy5TWV6MiFobD9QwdGqTqPPviy5z1aYIY9Gi6f5PDfoc1b8rQIDAQAB
AoGAPyYqvVksUj9Reh8jDukdoLrRltQxiqWfE70lQpUxkeuVCjibUJoQX/x86WT
h050yBb/tjb8qpKmwP8PlomlGnbkWF0qis0CBvmno+9hd0MVMv7f1EZb34UdNd
MLXGggoIY16K0YWAMfdWaoPtLA582aTOIA3PHURfLKi18YECQC75xvcmMk5nF2Y
mSMWn8zGa7YRXhWDNzM9qsikV503fngq/CmQ4dKOq2kslTs7CiAWJ2j9ppJqH
M1IRhr9AkeAu0/XJXYWBO+y7Hwgv69b3ml8N+OiklGtSLlT/gqzfxTIUzzukf
iygp/vebN2A8+00JAdhyX60Nuc0YiOBPCQAZpaq1g6P2pFhlH//ZUnH3oIONTs9
2Y41npSQyRfAt9t/qEpDN8NoWQGiBmoi7dud3T0EINW0dHZrsz6QLGX7QJAJeX
pNQ/biuk4NN5NevY4ZNA91uNs4+vThvugSx0Z78YCrSsYlTDxOVmc4hZUUISRj
mvPLSjbrI9W77/NIQJBAJ1x7rowwKdbSQ9NKO32sXInprZIKU57ws6mVxkFAQC
r/b0rEP9Z1eYS0i659ufjsPxTk+9zZAd0K5YKe9hTtl=
-----END RSA PRIVATE KEY-----
```

복구된 개인키

```
Public key from Private key:
-----BEGIN PUBLIC KEY-----
MIGfMA0GCsGSIb3DQEBAQUAA4GNADCBiQKBgQCFjGbu0B7qkbq33uksrwaHs0Wh
IAfzRTCj6LbOOVW8SRzkvj83Xaeu9oSSYolrS80z4GRDjzetzrhAA2qM8N53wCz
ZyqiW9/51zdo1BInQ/q5RGA83QPcENKZy5TWV6MiFobD9QwdGqTqPPviy5z1aY
Y9Gi6f5PDfoc1b8rQIDAQAB
-----END PUBLIC KEY-----
```

개인키로부터 복구된 공개키

## 3.3 인증기관에서 사용자 인증서 발급하기

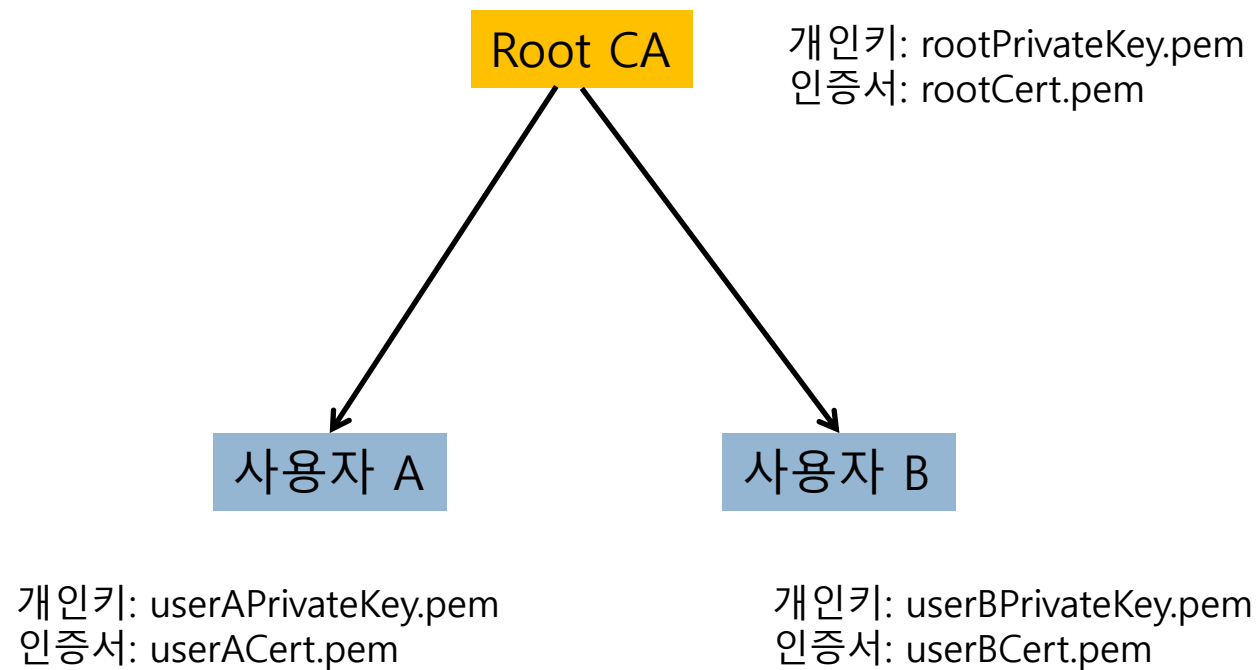
46

- 루트인증기관 준비(인증기관에서 자체서명인증서 생성)
  - ▣ 키쌍 생성, 개인키를 파일로 저장
  - ▣ 자체서명인증서 생성, 인증서를 파일로 저장
  
- 개인 사용자에게 인증서 발급
  - ▣ 개인의 키쌍 생성, 개인키를 파일로 저장 (클라이언트)
  - ▣ 인증기관에게 인증서 발급 요청 (클라이언트→인증기관)
  - ▣ 개인에게 인증서 발급 (인증기관→클라이언트)
  - ▣ 개인은 인증서를 파일로 저장 (클라이언트)

# 인증기관에서 사용자 인증서 발급하기

47

## □ 시나리오



# 인증서에서 필드 정보 읽어오기

48

```
console.log('Serial number: '+caCert.serialNumber);  
console.log('validity-notBefore: '+caCert.validity.notBefore);  
console.log('validity-notAfter: '+caCert.validity.notAfter);  
  
console.log('Common Name: '+caCert.subject.getField('CN').value);  
console.log('Organization: '+ caCert.subject.getField('O').value);  
console.log('Organization Unit: '+ caCert.subject.getField('OU').value);  
console.log('Locality Name: '+caCert.subject.getField('L').value);  
console.log('State Name: '+caCert.subject.getField('ST').value);  
console.log('Country Name: '+caCert.subject.getField('C').value);
```

```
Serial number: 01  
validity-notBefore: Thu Nov 16 2017 10:39:53 GMT+0900 (대한민국 표준시)  
validity-notAfter: Fri Nov 16 2018 10:39:53 GMT+0900 (대한민국 표준시)  
Common Name: Byoungcheon Lee  
Organization: Joongbu Univ.  
Organization Unit: Dept. of Information Security  
Locality Name: Goyang-si  
State Name: Gyeonggi-do  
Country Name: KR
```



# 개인 사용자에게 인증서 발급 (1/2)

49

cert.js

```
var forge = require('node-forge');
var fs = require('fs');
var pki = forge.pki;
var user = 'userB'; // 사용자명 설정

// 1. CA 개인키와 인증서를 파일에서 읽어오기
var caCertPem = fs.readFileSync('rootCert.pem', 'utf8');
var caPrivateKeyPem = fs.readFileSync('rootPrivateKey.pem', 'utf8');
var caCert = pki.certificateFromPem(caCertPem);
var caPrivateKey = pki.privateKeyFromPem(caPrivateKeyPem);
var verified = caCert.verify(caCert);
console.log('CA인증서의 유효성 검증: '+verified);

// -----
// 2. 사용자 키쌍 생성
var keys = pki.rsa.generateKeyPair(2048);

// 3. 사용자 개인키를 파일로 저장
console.log(pki.privateKeyToPem(keys.privateKey));
fs.writeFileSync(user+"PrivateKey.pem", pki.privateKeyToPem(keys.privateKey));
console.log('사용자 개인키 저장 - '+user+'PrivateKey.pem \n');

// 4. 사용자 인증서 객체 생성
var cert = pki.createCertificate();

cert.publicKey = keys.publicKey;
cert.serialNumber = '01';
cert.validity.notBefore = new Date();
cert.validity.notAfter.setFullYear(cert.validity.notBefore.getFullYear() + 1);
```

```
// 5. 사용자 정보 설정
var attrs = [{
  name: 'commonName', // 사용자명
  //shortName: 'CN',
  value: user
}, {
  name: 'countryName', // 국가
  //shortName: 'C',
  value: 'KR'
}, {
  name: 'stateOrProvinceName', // 주, 지역
  //shortName: 'RT',
  value: 'Gyeonggi-do'
}, {
  name: 'localityName', // 도시명
  //shortName: 'L',
  value: 'Goyang-si'
}, {
  name: 'organizationName', // 기관명
  //shortName: 'O',
  value: 'Joongbu Univ.'
}, {
  name: 'organizationalUnitName', // 부서명
  //shortName: 'OU',
  value: 'Dept. of Information Security'
}];
cert.setSubject(attrs);
```

# 개인 사용자에게 인증서 발급 (2/2)

50

```
// 6. CA 정보 설정. 인증서에서 읽어와서 자동 설정
var caAttrs = [{
  name: 'commonName', // shortName: 'CN',
  value: caCert.subject.getField('CN').value
}, {
  name: 'countryName', // shortName: 'C',
  value: caCert.subject.getField('C').value
}, {
  name: 'stateOrProvinceName', // shortName: 'ST',
  value: caCert.subject.getField('ST').value
}, {
  name: 'localityName', // shortName: 'L',
  value: caCert.subject.getField('L').value
}, {
  name: 'organizationName', // shortName: 'O',
  value: caCert.subject.getField('O').value
}, {
  name: 'organizationalUnitName', // shortName: 'OU',
  value: caCert.subject.getField('OU').value
}];
cert.setIssuer(caAttrs);
```

인증기관 issuer 정보 입력  
- caCert에서 읽어온 정보를 입력  
- 정보가 다를 경우 사용자인증서  
검증시 에러 발생

확장영역 정보 입력

```
// 7. 확장영역 설정
cert.setExtensions([{
  name: 'basicConstraints',
  cA: true
}, {
  name: 'keyUsage',
  keyCertSign: true,
  digitalSignature: true,
  nonRepudiation: true,
  keyEncipherment: true,
  dataEncipherment: true
}, {
  name: 'extKeyUsage',
  serverAuth: true,
  clientAuth: true,
  codeSigning: true,
  emailProtection: true,
  timeStamping: true
}, {
  name: 'nsCertType',
  client: true,
  server: true,
  email: true,
  objsign: true,
  sslCA: true,
  emailCA: true,
  objCA: true
}, {
  name: 'subjectAltName',
  altNames: [{
    type: 6, // URI
    value: 'http://example.org/webid#me'
  }, {
    type: 7, // IP
    ip: '127.0.0.1'
  }]
}, {
  name: 'subjectKeyIdentifier'
}]);
```

```
// 8. CA가 서명하여 사용자 인증서 생성
cert.sign(caPrivateKey); // CA 개인키로 서명
console.log('사용자 인증서 생성');
console.log(pki.certificateToPem(cert));
```

```
// 9. 사용자 인증서 검증
var verified = caCert.verify(cert);
console.log('사용자 인증서 검증: '+verified);
```

```
// 10. 사용자 인증서 저장
fs.writeFileSync(user+"Cert.pem", pki.certificateToPem(cert));
console.log('사용자 인증서 저장 - '+user+'Cert.pem');
```

인증서 파일 저장 - userCert.pem

# 인증서 검증

51

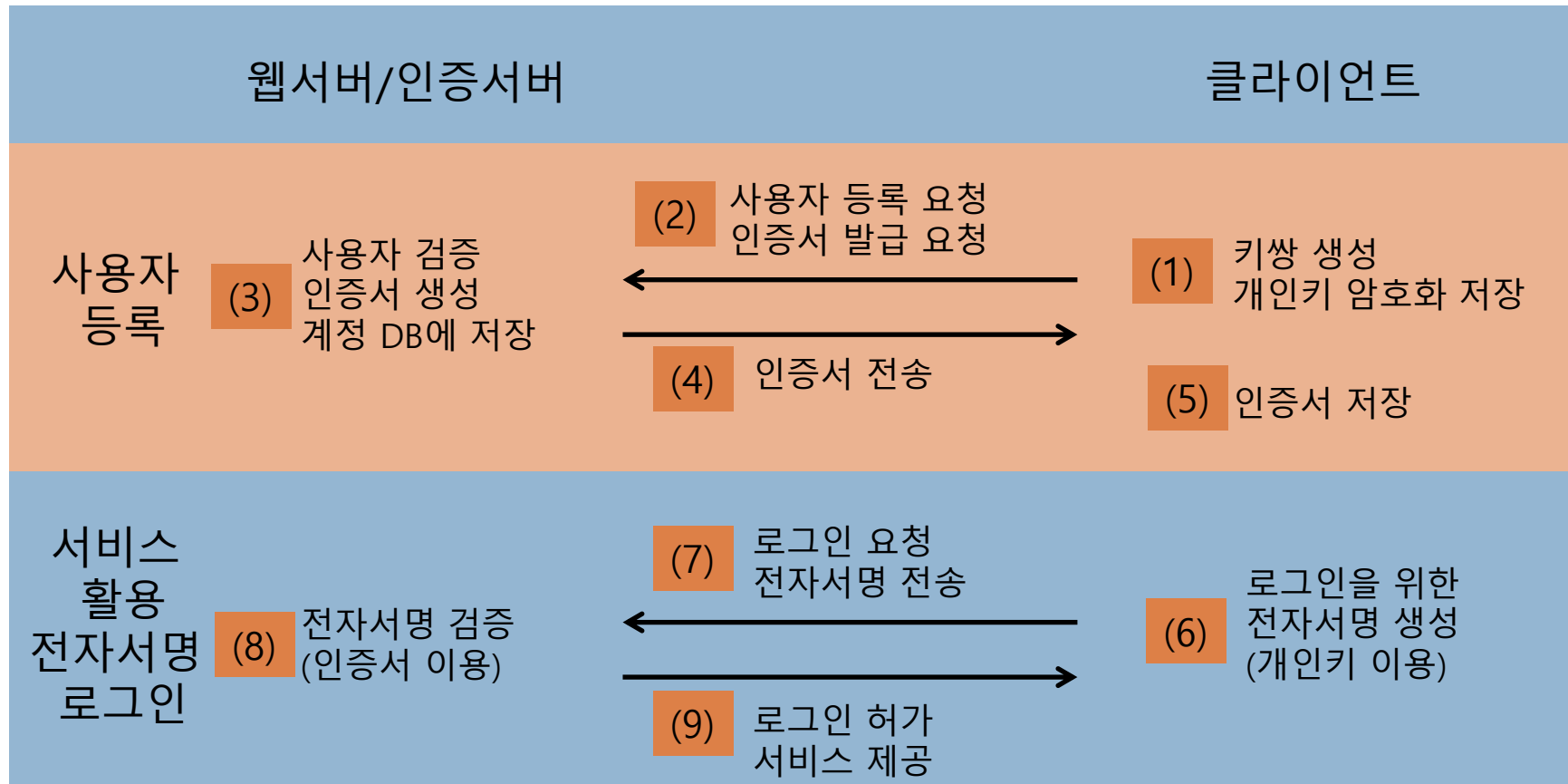
```
// 사용자 인증서 검증  
var verified = caCert.verify(cert);  
console.log('사용자 인증서 검증: '+verified);
```

CA인증서.verify(사용자인증서)

setIssuer 함수의 caAttrs 정보가 CA인증서 정보와 다를 경우  
사용자인증서 검증시 에러 발생

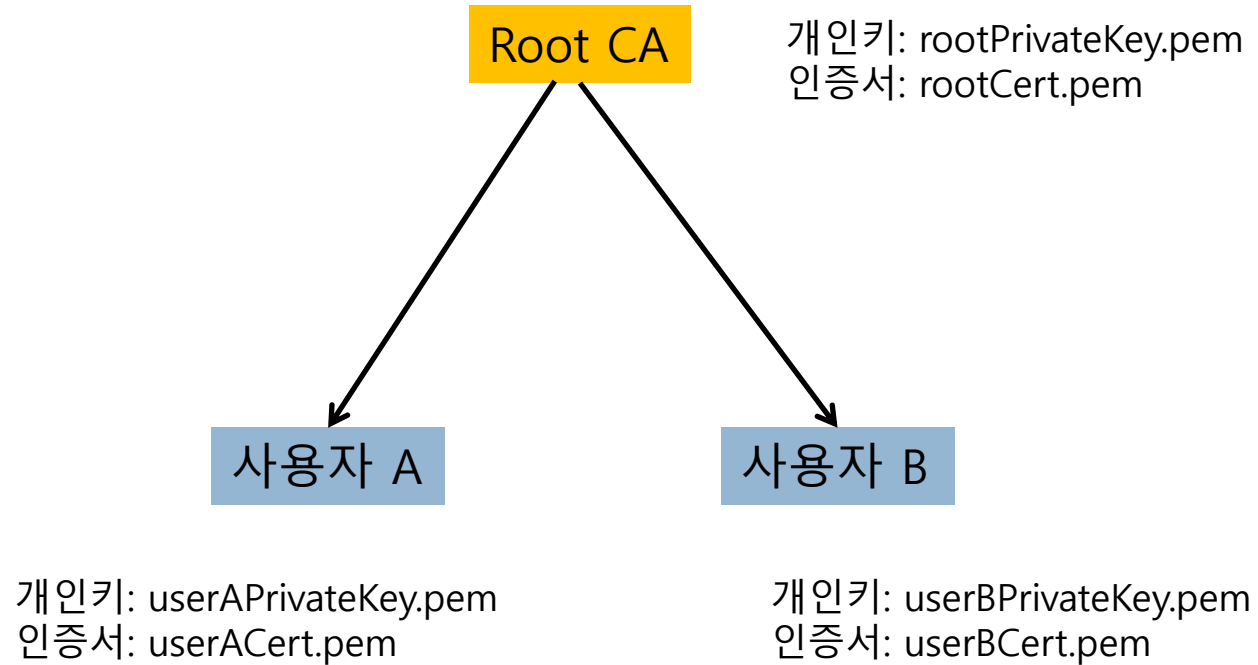
## 3.4 인증서 활용

52



# 인증서 활용 시나리오

53



1. 전자서명 로그인
2. 서명된 메시지 전송
3. 전자봉투 전송

# 전자서명 로그인

54

```
var forge = require('node-forge');
var fs = require('fs');
var pki = forge.pki;

// 1. 사용자 A가 서버에게 서명된 로그인 요청 메시지를 전송

// 1.1 사용자 A의 개인키 읽어옴
var userAPrivateKeyPem = fs.readFileSync('userAPrivateKey.pem', 'utf8');
var userAPrivateKey = pki.privateKeyFromPem(userAPrivateKeyPem);

// 1.2 서명된 로그인 메시지 생성
var ID = 'userA';
var time = new Date().getTime();

var md = forge.md.sha1.create();
md.update(ID+time, 'utf8');
var signature = userAPrivateKey.sign(md);
var signatureHex = forge.util.bytesToHex(signature);

// 1.3 <ID, time, signature>를 로그인 메시지로 서버에 전송
console.log('사용자 A측');
console.log('ID: '+ID);
console.log('현재시간: '+time);
console.log('Signature: '+signatureHex);
```

사용자 A가 서버에게  
서명된 로그인 요청 메시지 전송

// 2. 서버가 사용자의 로그인 요청 메시지 검증

// 2.1 사용자 A로부터 <ID, time, signature>를 수신  
console.log('서버측');

// 2.2 사용자 계정 DB로부터 사용자 A의 인증서 읽어옴  
var userACertPem = fs.readFileSync('userACert.pem', 'utf8');  
var userACert = pki.certificateFromPem(userACertPem);

// 2.3 인증기관의 인증서 읽어옴  
var caCertPem = fs.readFileSync('rootCert.pem', 'utf8');  
var caCert = pki.certificateFromPem(caCertPem);

// 2.4 사용자 A의 인증서 검증  
var verified1 = caCert.verify(userACert);  
console.log('사용자 A의 인증서 검증: '+verified1);

// 2.5 전자서명 검증  
var userAPublicKey = userACert.publicKey;  
var md = forge.md.sha1.create();  
md.update(ID+time, 'utf8');  
var verified2 = userAPublicKey.verify(md.digest().bytes(), signature);  
console.log('서명의 유효성 검증: '+verified2);

var timeS = new Date().getTime();  
console.log('서버 현재시간: '+timeS);  
var timeDiff = timeS - time;  
console.log('현재시간의 유효성 검증: '+timeDiff/1000+'초');  
var verified3;  
if(timeDiff < 1000) verified3 = true;

```
if(verified1 === true && verified2 === true && verified3 === true) {  
  console.log('로그인 허용');  
} else {  
  console.log('로그인 거부');  
}
```

# 서명된 메시지 전송

55

## 송신자 A가 수신자 B에게 서명된 메시지 전송

```
var forge = require('node-forge');
var fs = require('fs');
var pki = forge.pki;

var userA = 'userA'; // 사용자명 설정 (송신자 userA)
var userB = 'userB'; // 사용자명 설정 (수신자 userB)

// 1. 송신자 A가 수신자 B에게 서명된 메시지 전송

// 1.1 송신자 A의 개인키 읽어옴
var userAPrivateKeyPem = fs.readFileSync('userAPrivateKey.pem', 'utf8');
var userAPrivateKey = pki.privateKeyFromPem(userAPrivateKeyPem);

// 1.2 송신자 A: 메시지와 전자서명 생성
var message = 'Hello world. 안녕하세요.';
var md = forge.md.sha1.create();
md.update(message, 'utf8');
var signature = userAPrivateKey.sign(md);

// 1.3 <메시지, 전자서명>을 수신자 B에게 전송
console.log('송신자 A측');
console.log('Message: ' + message);
console.log('Signature: ' + forge.util.bytesToHex(signature));
```

```
// 2. 수신자 B의 서명 검증

// 2.1 송신자 A로부터 <메시지, 전자서명>을 수신
console.log('수신자 B측');

// 2.2 송신자 A의 인증서 읽어옴
var userACertPem = fs.readFileSync('userACert.pem', 'utf8');
var userACert = pki.certificateFromPem(userACertPem);

// 2.3 인증기관의 인증서 읽어옴
var caCertPem = fs.readFileSync('rootCert.pem', 'utf8');
var caCert = pki.certificateFromPem(caCertPem);

// 2.4 송신자 A의 인증서 유효성 검증
var verified = caCert.verify(userACert);
console.log('송신자 A의 인증서 검증: ' + verified);

// 2.5 전자서명 검증
var userAPublicKey = userACert.publicKey;
var md = forge.md.sha1.create();
md.update(message, 'utf8');
var verified = userAPublicKey.verify(md.digest().bytes(), signature);
console.log('서명의 유효성 검증: ' + verified);
```

# 전자봉투(Digital Envelope) 전송

56

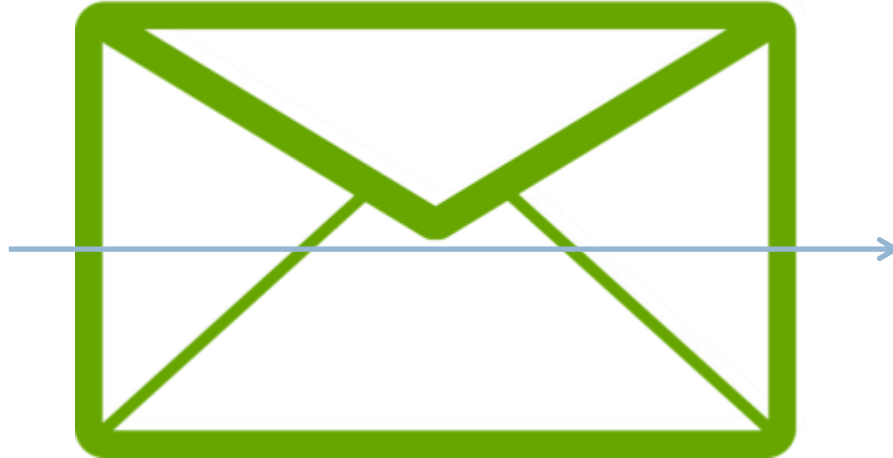
## □ 3가지 기능을 한번에

- 전자서명 : 송신자의 개인키로 메시지에 전자서명
- 메시지 암호화 : 난수 세션키로 메시지와 서명값을 암호화
- 세션키 전달 : 세션키를 수신자의 공개키로 암호화



송신자 A

1. 전자서명 생성
2. 메시지 암호화
3. 세션키 전달



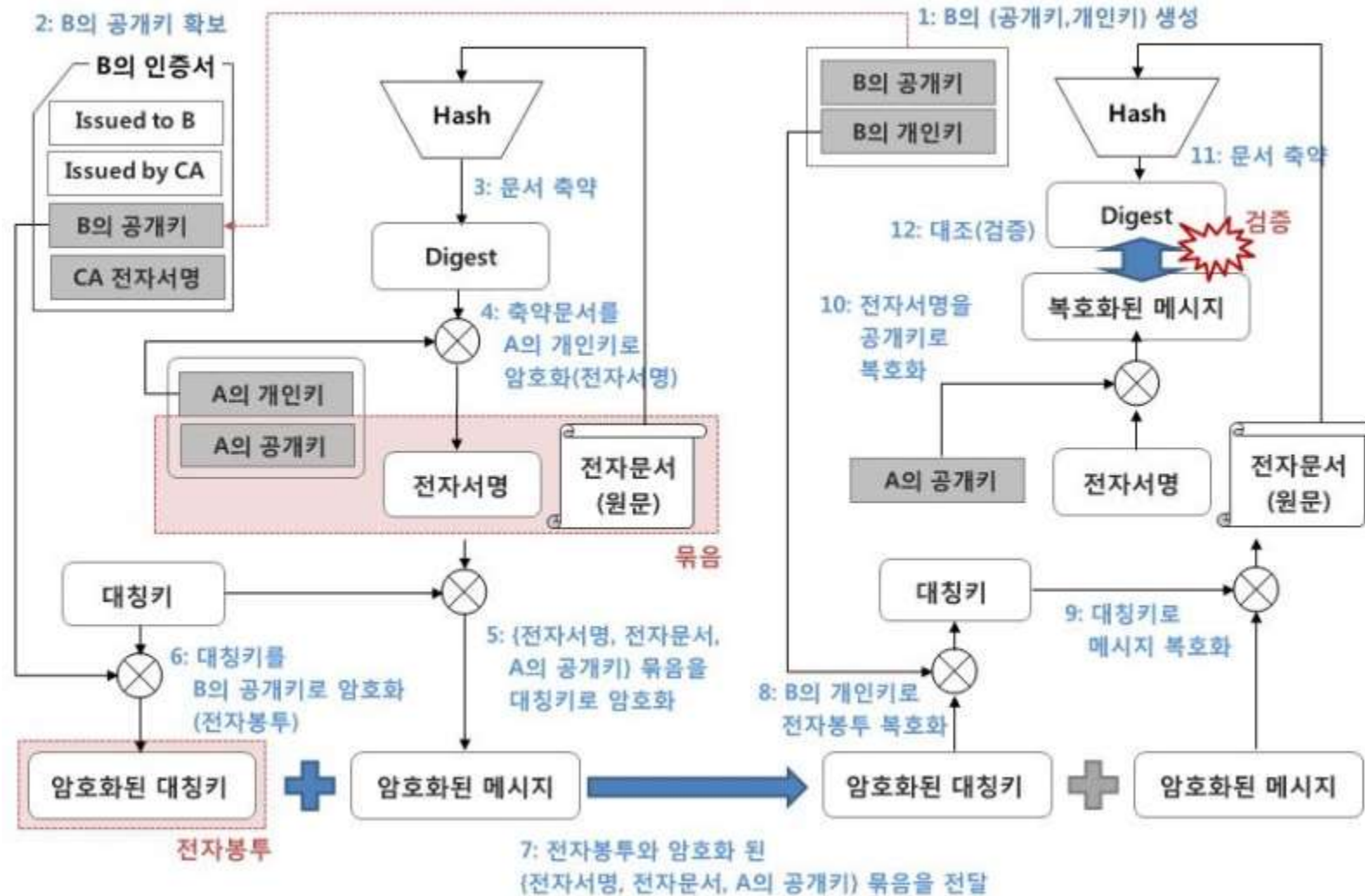
수신자 B

1. 세션키 복구
2. 메시지 복호화
3. 전자서명 검증



# 전자봉투(Digital Envelope) 전송

57



서명 후 암호화

복호화 후 서명 검증

# 전자봉투 전송

58

```
var forge = require('node-forge');  
var fs = require('fs');  
var pki = forge.pki;  
  
var userA = 'userA'; // 사용자명 설정 (송신자 userA)  
var userB = 'userB'; // 사용자명 설정 (수신자 userB)  
  
// 전자봉투 생성  
// 1. 송신자의 개인키로 메시지에 전자서명 생성  
// 2. 메시지와 서명값을 난수 세션키로 암호화  
// 3. 난수 세션키를 수신자의 공개키로 암호화  
  
// 전자봉투 열기  
// 1. 수신자의 개인키로 복호화하여 난수 세션키 복구  
// 2. 난수 세션키로 복호화하여 메시지와 서명값을 복구  
// 3. 송신자의 공개키로 전자서명 검증
```

# 전자봉투 전송 - 송신자 A

59

```
// 1. 송신자 A가 수신자 B에게 전자봉투 전송
//-----

// 1.1 송신자 A의 개인키 읽어옴
var userAPrivateKeyPem = fs.readFileSync('userAPrivateKey.pem', 'utf8');
var userAPrivateKey = pki.privateKeyFromPem(userAPrivateKeyPem);

// 1.2 수신자 B의 인증서 읽어옴
var userBCertPem = fs.readFileSync('userBCert.pem', 'utf8');
var userBCert = pki.certificateFromPem(userBCertPem);
var userBPublicKey = userBCert.publicKey;

// 1.3 인증기관의 인증서 읽어옴
var caCertPem = fs.readFileSync('rootCert.pem', 'utf8');
var caCert = pki.certificateFromPem(caCertPem);

// 1.4 수신자 B의 인증서 유효성 검증
var verifiedB = caCert.verify(userBCert);
console.log('1.4 수신자 B의 인증서 검증: '+verifiedB);

// 1.5 송신자 A: 메시지에 대한 전자서명 생성
var message = 'Hello world. 안녕하세요.';
var md = forge.md.sha1.create();
md.update(message, 'utf8');
var signature = userAPrivateKey.sign(md);
var signatureHex = forge.util.bytesToHex(signature);
var messageObject = {
  msg: message,
  sigHex: signatureHex
};
var messageString = JSON.stringify(messageObject);
console.log('1.5 JSON Message string: \n'+messageString);
```

```
// 1.6 송신자 A: 세션키로 메시지+전자서명을 암호화
var keySize = 16; // 16 => AES-128, 24 => AES-192, 32 => AES-256
var ivSize = 16;
var key = forge.random.getBytesSync(keySize);
var iv = forge.random.getBytesSync(ivSize);
var keyObject = {
  key: key,
  iv: iv
}
var keyString = JSON.stringify(keyObject);
console.log('1.6 JSON key string: \n'+keyString);

var someBytes = forge.util.encodeUtf8(messageString);
var cipher = forge.cipher.createCipher('AES-CBC', key);
cipher.start({iv: iv});
cipher.update(forge.util.createBuffer(someBytes));
cipher.finish();
var encrypted = cipher.output;
var encryptedMessageHex = forge.util.bytesToHex(encrypted);
console.log('1.6 encryptedMessageHex: \n'+encryptedMessageHex);

// 1.7 세션키를 수신자 B의 공개키로 암호화
// console.log('RSA-OAEP');
var encryptedSessionKey = userBPublicKey.encrypt(keyString, 'RSA-OAEP');
var encryptedSessionKeyHex = forge.util.bytesToHex(encryptedSessionKey);
console.log('1.7 encryptedSessionKeyHex: \n'+encryptedSessionKeyHex);
```

# 전자봉투 전송 - 수신자 B

60

```
// 송신자 A가 수신자 B에게 전송하는 메시지
// <encryptedMessageHex, encryptedSessionKeyHex>

// 2. 수신자 B가 수신한 전자봉투를 처리
// -----

// 2.1 수신자 B의 개인키 읽어옴
var userBPrivateKeyPem = fs.readFileSync('userBPrivateKey.pem', 'utf8');
var userBPrivateKey = pki.privateKeyFromPem(userBPrivateKeyPem);

// 2.2 송신자 A의 인증서 읽어옴
var userACertPem = fs.readFileSync('userACert.pem', 'utf8');
var userACert = pki.certificateFromPem(userACertPem);

// 2.3 인증기관의 인증서 읽어옴
var caCertPem = fs.readFileSync('rootCert.pem', 'utf8');
var caCert = pki.certificateFromPem(caCertPem);

// 2.4 송신자 A의 인증서 유효성 검증
var verifiedA = caCert.verify(userACert);
console.log('2.4 송신자 A의 인증서 검증: '+verifiedA);

// 2.5 세션키를 복구
var encryptedSessionKey = forge.util.hexToBytes(encryptedSessionKeyHex);
var decryptedKeyString = userBPrivateKey.decrypt(encryptedSessionKey, 'RSA-OAEP');
console.log('2.5 Decrypted session key string: \n'+decryptedKeyString);
var keyObject1 = JSON.parse(decryptedKeyString);
var key1 = keyObject1.key;
var iv1 = keyObject1.iv;
```

```
// 2.7 세션키로 메시지 복호화
var encryptedMessage = forge.util.hexToBytes(encryptedMessageHex);
var decipher = forge.cipher.createDecipher('AES-CBC', key1);
decipher.start({iv: iv1});
decipher.update(forge.util.createBuffer(encryptedMessage));
var result = decipher.finish(); // check 'result' for true/false
var messageString1 = decipher.output;
console.log('2.7 result: \n'+messageString1 );
var messageObject1 = JSON.parse(messageString1);
var message1 = messageObject1.msg;
var signatureHex1 = messageObject1.sigHex;
var signature1 = forge.util.hexToBytes(signatureHex1);

// 2.8 전자서명 검증
var userAPublicKey = userACert.publicKey;
var md = forge.md.sha1.create();
md.update(message1, 'utf8');
var verified = userAPublicKey.verify(md.digest().bytes(), signature1);
console.log('2.8 서명의 유효성 검증: '+verified);
```

# JSON (Javascript Object Notation)

61

```
var messageObject = {  
  msg: message,  
  sigHex: signatureHex  
};
```

JSON.stringify()

```
var messageString =  
{"msg": " real message...",  
"sigHex": "real signature..."}
```

JSON.parse()

## 4. 공개키기반구조(PKI)



62

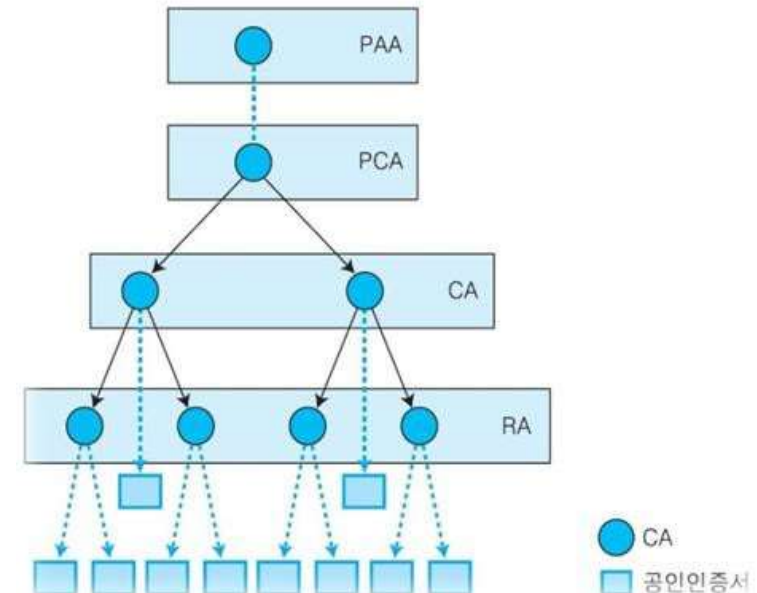
- **공개키기반구조(public key infrastructure, PKI)**
  - ▣ 공개키 암호 방식을 바탕으로 한 인증서를 활용하는 소프트웨어, 하드웨어, 사용자, 정책 및 제도 등을 총칭하는 용어
  - ▣ 공개키암호기술이 안전하게 사용될 수 있는 환경을 제공
  
- **인증서 표준**
  - ▣ X.509, The Directory: Authentication Framework, 1993.
  
- **공개키기반구조 표준**
  - ▣ PKIX: Internet X.509 Public Key Certificate Infrastructure.
  - ▣ <https://datatracker.ietf.org/wg/pkix/charter/>

## □ 인증기관의 계층구조

- ▣ 정책승인기관(PAA, Policy approval authority)
- ▣ 정책인증기관(PCA, policy certification authority)
- ▣ 인증기관(CA, certification authority)
  - 인증서를 발급하는 기관
- ▣ 등록기관(RA, registration authority)
  - 사용자의 신분을 확인하고 등록요청을 대행하는 기관
  - 사용자 대면 신분인증 필요

화살표는 인증서  
발급관계를 나타냄

상위 인증기관이  
하위 인증기관의  
인증서를 발급

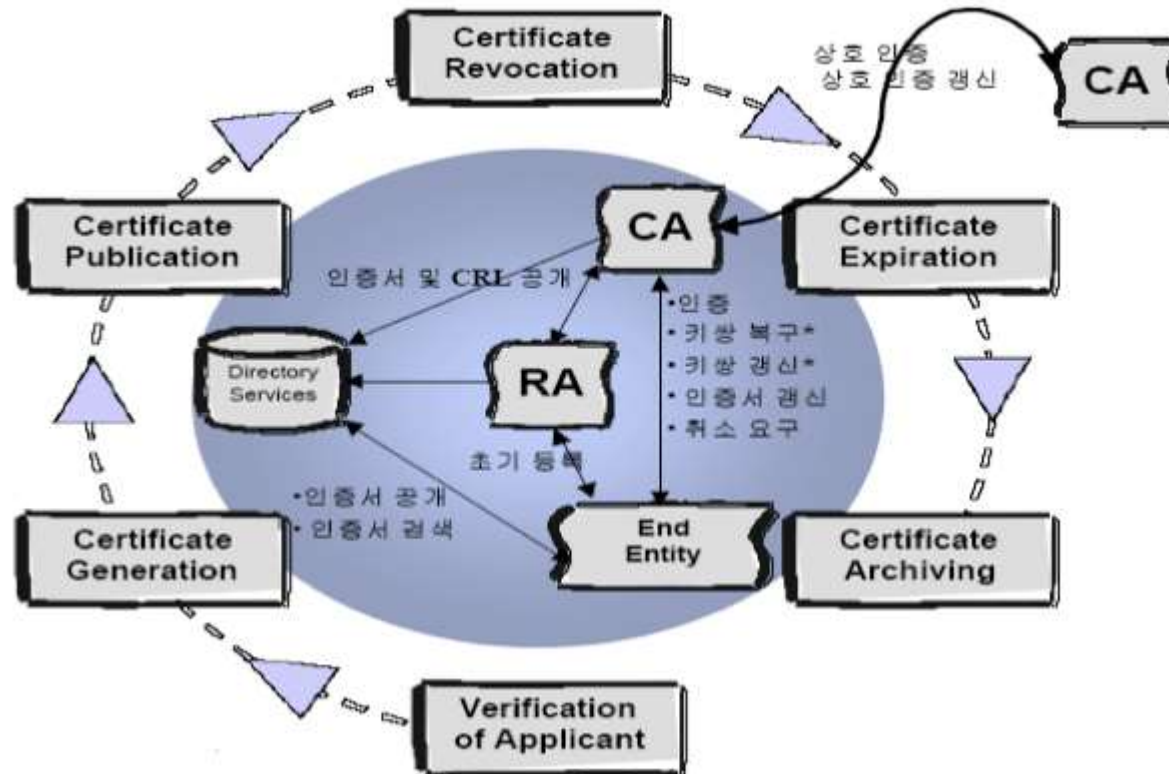


# 인증기관

64

## □ 인증기관의 업무

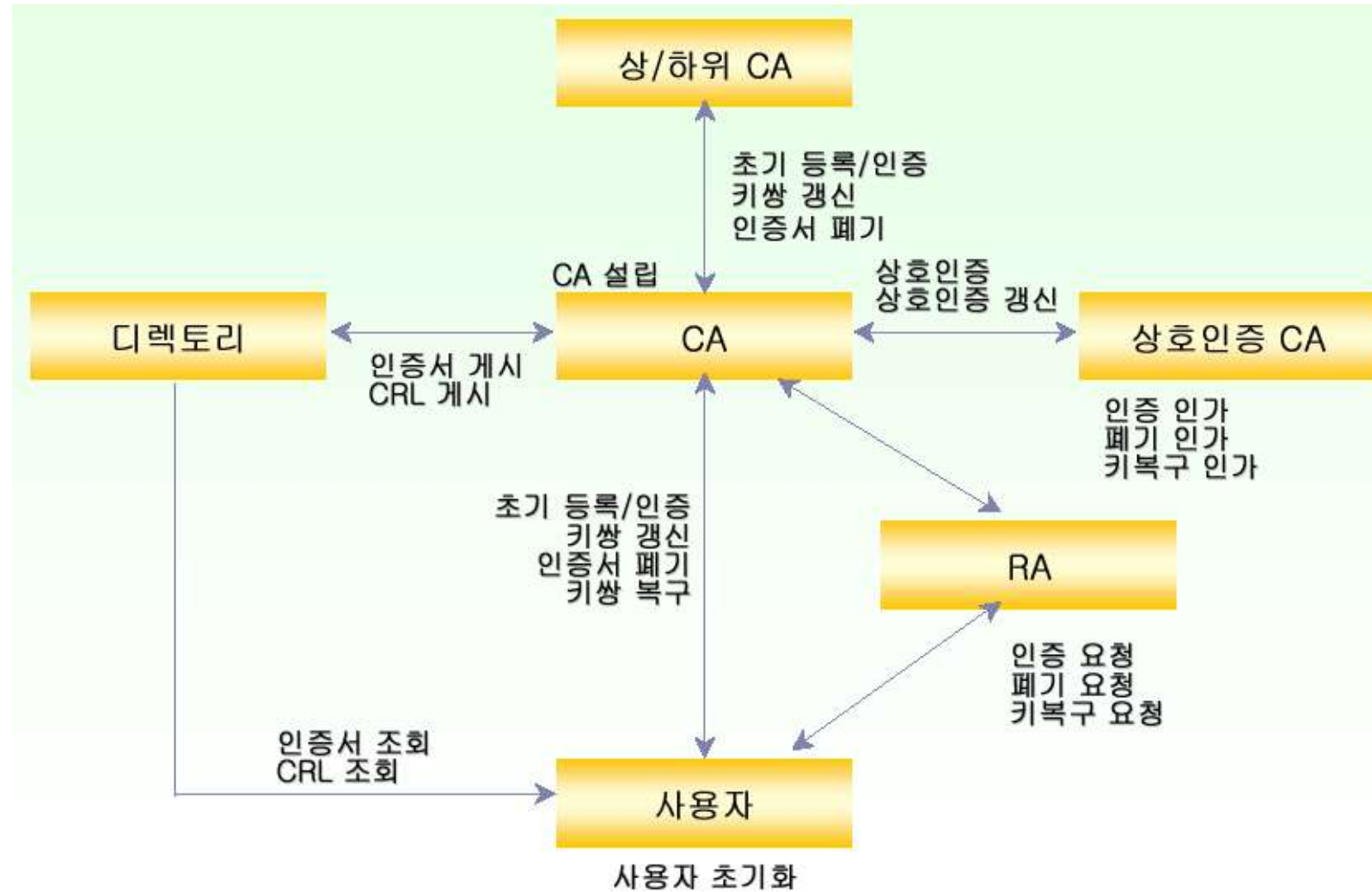
- 인증서 발급
- 인증서 관리
- 인증서 배포
- 인증서 사용
- 인증서 저장
- 인증서 취소





# PKI 서비스 흐름도

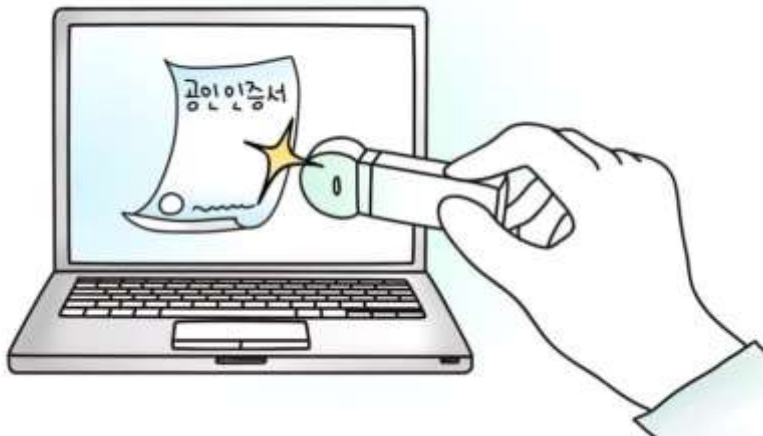
65



# 공인인증서

66

- 공인인증서
  - ▣ 공인인증기관이 발행하는 인증서
  - ▣ 널리 호환되어 이용 가능



# 공인인증서

67

## □ 공인인증기관

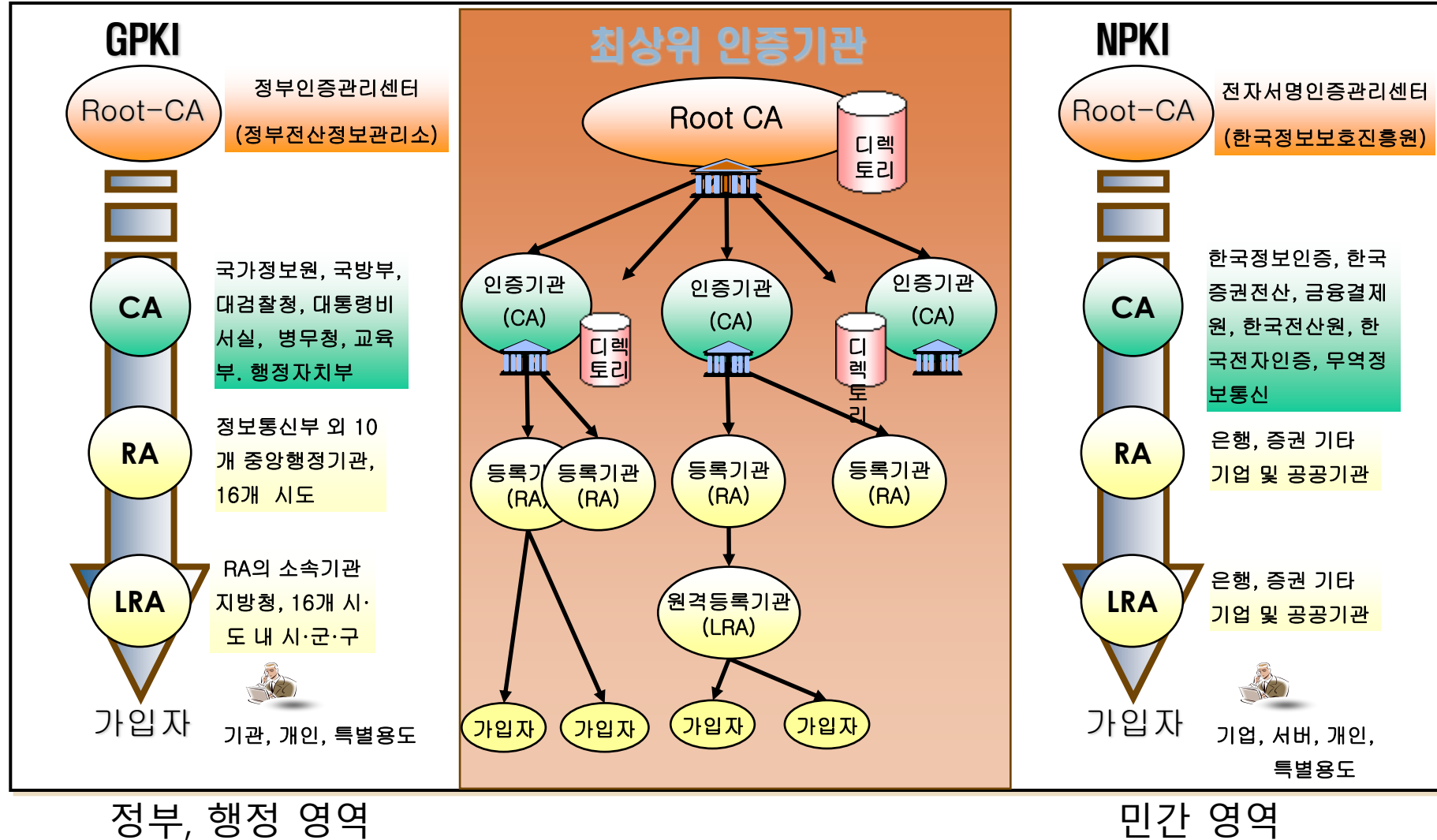
[표] 국내 공인인증기관

공인인증기관	공인인증기관 지정 일자	설립목적	특기사항
금융결제원	지정일: 2000년 4월 설립일: 1986년 6월	은행간 결제	비영리기관
코스콤	지정일: 2000년 2월 설립일: 1977년 9월	증권분야 전산인프라 구축	-
한국무역정보통신	지정일: 2002년 3월 설립일: 1991년 12월	무역업무자동화	-
한국전자인증	지정일: 2001년 11월 설립일: 1999년 3월	공인인증서비스	2000년 1월 (글로벌서비스)
한국정보인증	지정일: 2000년 2월 설립일: 1999년 7월	공인인증서비스	-

자료: 아이투자, 한국전자인증 분기보고서

# 공인인증체계

68



# 전자정부에서의 활용

69

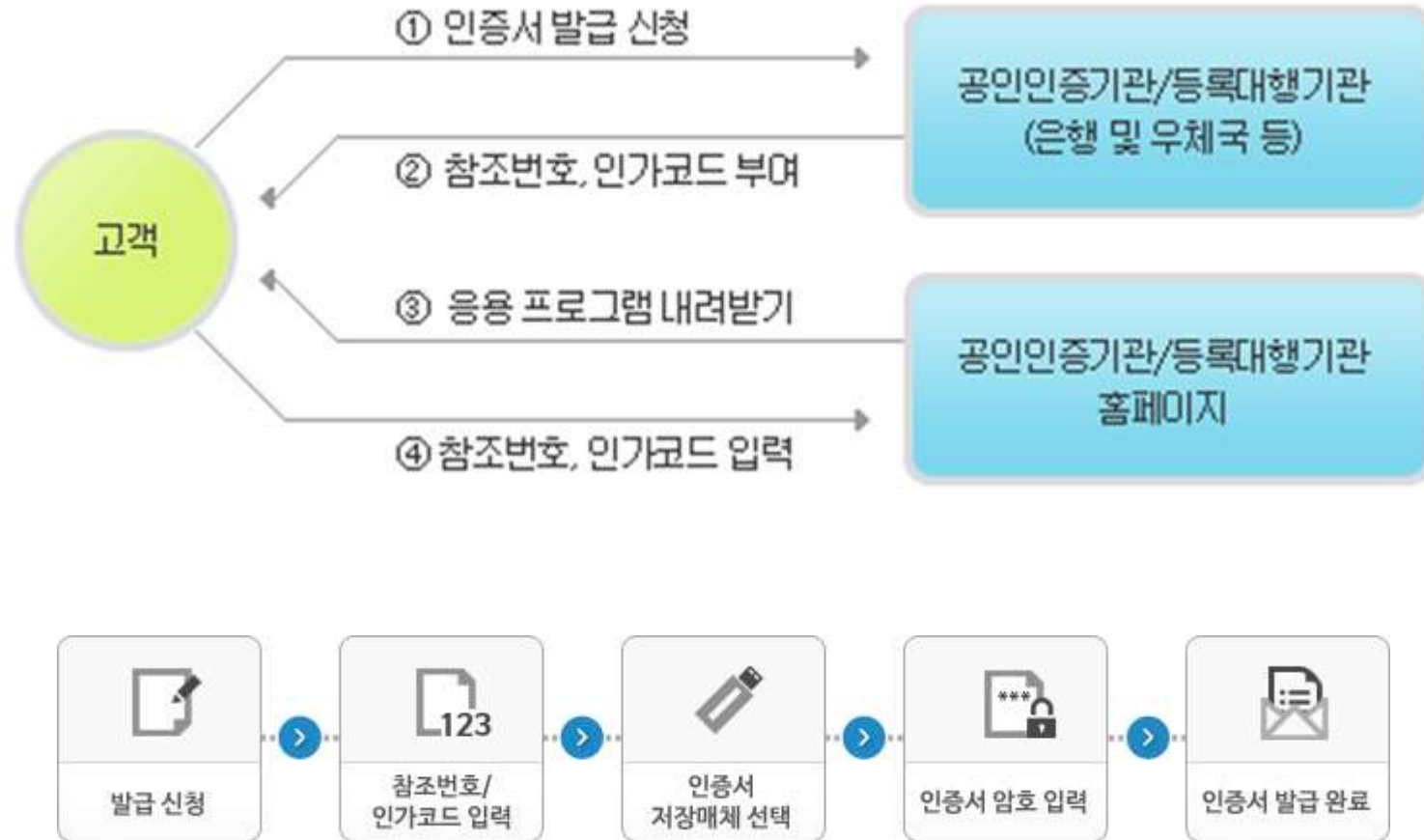


행정자치부	교육부	조달청	특허청
전자민원 (G4C)	교육행정정보서비스 (NEIS)	전자조달시스템 (G2B)	특허넷시스템

인증 시스템		인증센터 구축	RA	X	CA
인증서 사용	행정	전자관인	공인인증(교육부용)	전자관인, 공인인증	사설인증, 전자관인
	대민	공인인증	공인인증	공인인증서	사설인증서, 공인인증서
인증·보안 적용		전자결재 민원신청서 적용	행정업무 중요 정보의 암호화, 전자서명	입찰업무 전자입찰서 (XML 기반)	행정업무 전자출원 적용
운영		센터 운영	RA 이중화	X	사설 CA시스템
인원		운용관리(10~20명)	지역교육청 전산실 담당자	RA 운영인원(X)	운영인원 필요

# 공인인증서 발급 절차

70



# 공인인증서

71

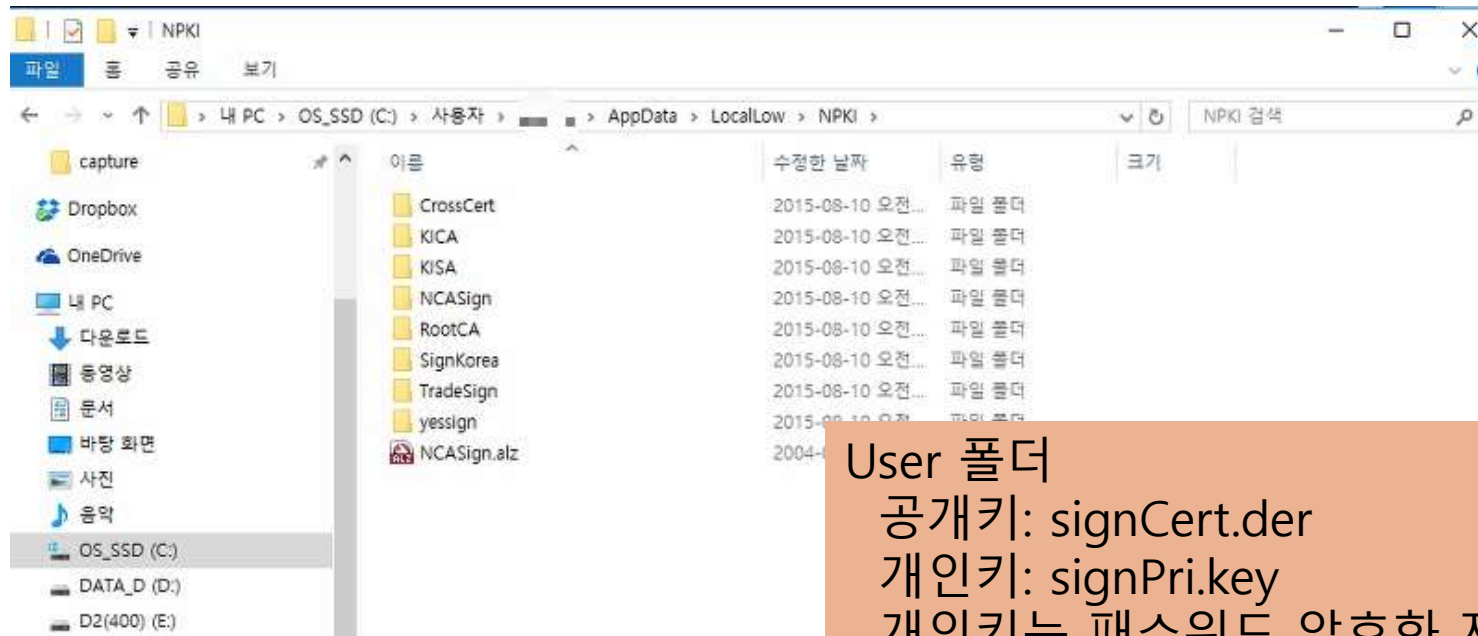
## □ 공인인증서의 저장 위치

### ▣ Windows XP

- C: > Program files > NPki

### ▣ Windows 7 이후

- C: > 사용자 > 사용중인 계정 이름 > AppData > LocalLow > NPki



# 공인인증서 정보 보기

72

The screenshot shows a window titled '인증서' (Certificate) with tabs for '일반' (General), '자세히' (Details), and '인증 경로' (Certification Path). The '일반' tab is active, displaying a table of certificate fields and their values. Below the table, the Distinguished Name (DN) is shown as 'CN = yessignCA Class 2, OU = AccreditedCA, O = yessign, C = kr'. At the bottom, there are buttons for '속성 편집(E)...' (Edit Properties), '파일에 복사(C)...' (Copy to File), and '확인' (OK).

필드	값
버전	V3
일련 번호	1e d1 d7 ce
서명 알고리즘	sha256RSA
서명 해시 알고리즘	sha256
발급자	yessignCA Class 2, AccreditedCA
유효 기간(시작)	2017년 3월 3일 금요일 오...
유효 기간(끝)	2018년 3월 18일 일요일 오...
주체	이병천(Byoungcheon Lee)0...

CN = yessignCA Class 2  
OU = AccreditedCA  
O = yessign  
C = kr

속성 편집(E)...    파일에 복사(C)...    확인

서명 해시 알고리즘: sha256  
공개키: RSA 2048 bits  
키 사용: 전자서명, 부인방지  
인증서 정책:

<http://www.yessign.or.kr/cps.htm>

The screenshot shows the '인증업무준칙(CPS)' (Certification Business Rules) page. It includes a header with the title and a sub-header 'yessign 인증업무준칙에 대해 안내해 드립니다.' (We will guide you about the yessign certification business rules). Below this is a section with a document icon and text explaining the purpose of the rules. A '전문 다운로드' (Download Manual) button is present. The main content is organized into a grid of sections:

- 01 개요** (Overview): 1.1 비준 및 목적, 1.2 준칙의 명칭, 1.3 공인인증서발행인용처에 관한자, 1.4 준칙의 관리, 1.5 절차 및 처리
- 02 공인인증서 종류 및 수수료** (Types and Fees of Public Certificate): 2.1 공인인증서 종류, 2.2 공인인증업무 수수료
- 04 공인인증업무 관련정보의 광고** (Advertising of Information Related to Public Certificate Business): 4.1 광고 실비, 4.2 광고 방법
- 05 공인인증업무 시설 및 장비 보호조치** (Facility and Equipment Protection Measures for Public Certificate Business): 5.1 물리적 보호조치, 5.2 절차적 보호조치, 5.3 기술적 보호조치, 5.4 인적 보안, 5.5 접근사 기록



# 인증서 폐기

73

## □ 생성한 전자문서를 폐기할 수 있는가?

- ▣ 전자문서는 여러 곳에 복사 가능
- ▣ 사생활, 명예훼손
- ▣ 잊혀질 권리



## □ 발급한 인증서를 무효화시킬 수 있는가?

- ▣ 인증서 분실로 사용이 불가능해진 경우 재발급 필요
- ▣ 사용자의 신분이 변경되어 더 이상 사용하지 못하는 경우

# WANTED! 공개수배전단

74



『여러분의 신고와 제보가 사건해결의 결정적 단서가 됩니다』

## 사람을 찾습니다

**사건개요**

우병우 전 민정수석이 국조특위에 증인으로 채택되었으나 청문회 출석 요구서를 피해 이리저리 도망 다니고 있음. 집에는 개 짖는 소리만 나고 있음. 강도, 강박, 강모, 골프장에 찾아 갔으나 행방을 찾을 수 없었음.



**인상학적**

우병우(남, 50대) 키 175센티미터, 둥그런 얼굴에 안경을 썼고, 28가르파 머리, 앞수가 적고 팔장을 잘 끼고 눈을 잘 흘린다. 변병을 잘하고, 아들은 보너스를 매우 잘하고 청강이라는 회사는 유령들만 다닌다.

소재를 알고 계시거나 목격하신분들은 연락바랍니다. 청문회가 안타깝게 기다립니다.

**국회 경호실 02-788-2114 내선번호 2290**

## 대한민국 국회의장

국회 경호실 02-788-2114 내선번호 2290

국회 경호실 02-788-2114 내선번호 2290

# 인증서 폐기 목록

75

- Certificate Revocation List (CRL)
  - ▣ 해지되었거나 더 이상 유효하지 않은 인증서의 목록을 인증기관이 서명한 문서
  - ▣ CRL에 포함된 인증서는 유효하지 않으므로 신뢰해서는 안됨
  
- RFC 5280으로 표준화
  - ▣ Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile

# 인증서 폐기 목록

76

- 인증서취소목록(CRL)의 기본 영역
  - ▣ 서명 알고리즘 : CRL에 서명한 서명 알고리즘 ID 및 관련 데이터
  - ▣ 발급자 : 발급자 CA의 X.509 이름
  - ▣ 최근 수정 일자 : 최근 수정 일자(UTC Time)
  - ▣ 차후 수정 일자 : 다음 수정 일자(UTC Time)
  - ▣ 취소 인증서 목록 : 취소된 인증서 목록들
  - ▣ CRL확장자 : CRL 확장자 유무 및 내용
  - ▣ 발급자 서명문 : 발급자의 서명문

CRL의 확장 영역 : 기본 확장자 + 개체 확장자

기본 확장자

CA 키 고유 번호 : CRL에 서명한 키 번호

발급자 대체 이름 : CRL 발급자의 대체 이름(e-mail, IP 주소 등)

CRL 발급번호 : CRL에 대한 일련 번호

발급 분배점 : CRL 분배점 이름

델타 CRL지시자 : 최근에 취소된 목록만을 저장한 델타 CRL 지시자

개체 확장자

취소 이유 부호 : 인증서가 취소된 이유

명령 부호 : 해당 인증서를 만났을 경우 취해져야 할 명령

무효화 날짜 : 해당 인증서가 무효화된 날짜

인증서 발급자 : 간접 CRL에서의 해당 인증서 발급자

# 인증서 폐기 목록

77

- 인증서 폐기 이유 명시 (CRL reason code)
  - ▣ unspecified (0)
  - ▣ keyCompromise (1)
  - ▣ CACompromise (2)
  - ▣ affiliationChanged (3)
  - ▣ superseded (4)
  - ▣ cessationOfOperation (5)
  - ▣ certificateHold (6)
  - ▣ removeFromCRL (8)
  - ▣ privilegeWithdrawn (9)
  - ▣ ACompromise (10)

