以太坊的地址，代表一个账户，每个地址在以太坊的网络中都是唯一的。形如：0xCAb8eEA4799c21379c20eF5Baa2CC8aF1bEC475B
以太坊官方命令行客户端geth中，跟账户相关的命令是geth account，执行新建一个账户的命令：geth account new，输入密码成功生成账户时会返回这个账户的地址。
以太坊是如何生成这串在全网唯一的字符串？我们从geth程序的入口文件 cmd/geth/main.go入手，来分析下其源码：

cmd/geth/main.go

```
func init() {
    // Initialize the CLI app and start Geth
    app.Action = geth
    ...
    app.Commands = []cli.Command{
        ...
        // See accountcmd.go:
        accountCommand,
```

cmd/geth/accountcmd.go

```
    {
        Name:   "new",
        Usage:  "Create a new account",
        Action: utils.MigrateFlags(accountCreate),
        ...
        Description: `geth account new
        Creates a new account and prints the address....`,
    },
```

账户相关的命令在cmd/geth/accountcmd.go里，新建账户命令为new，其会调accountCreate方法，我们继续看：

```
cmd/geth/accountcmd.go

// accountCreate creates a new account into the keystore defined by the CLI flags.
func accountCreate(ctx *cli.Context) error {
    cfg := gethConfig{Node: defaultNodeConfig()}
    // Load config file.
    if file := ctx.GlobalString(configFileFlag.Name); file != "" {
        if err := loadConfig(file, &cfg); err != nil {
            utils.Fatalf("%v", err)
        }
    }
    utils.SetNodeConfig(ctx, &cfg.Node)
    scryptN, scryptP, keydir, err := cfg.Node.AccountConfig()

    if err != nil {
        utils.Fatalf("Failed to read configuration: %v", err)
    }

    password := getPassPhrase("Your new account is locked with a password.
    Please give a password. Do not forget this password.", true, 0, utils.MakePasswordList(ctx))

    address, err := keystore.StoreKey(keydir, password, scryptN, scryptP)

    if err != nil {
        utils.Fatalf("Failed to create account: %v", err)
    }
    fmt.Printf("Address: {%x}\n", address)
    return nil
}
```

代码比较清晰，获取配置并解析用户的密码，然后我们进入keystore.StoreKey看看，它会调用storeNewKey()来创建一个新的账户，具体表现为生成一对公私钥，再由私钥算出地址并构建一个自定义的Key，代码如下：

```
accounts/keystore/keystore_passphrase.go

// StoreKey generates a key, encrypts with 'auth' and stores in the given directory
func StoreKey(dir, auth string, scryptN, scryptP int) (common.Address, error) {
    _, a, err := storeNewKey(&keyStorePassphrase{dir, scryptN, scryptP}, crand.Reader, auth)
    return a.Address, err
}


accounts/keystore/key.go
func newKey(rand io.Reader) (*Key, error) {
    privateKeyECDSA, err := ecdsa.GenerateKey(crypto.S256(), rand)
    if err != nil {
        return nil, err
    }
    return newKeyFromECDSA(privateKeyECDSA), nil
}

func newKeyFromECDSA(privateKeyECDSA *ecdsa.PrivateKey) *Key {
    id := uuid.NewRandom()
    key := &Key{
        Id:         id,
        Address:    crypto.PubkeyToAddress(privateKeyECDSA.PublicKey),
        PrivateKey: privateKeyECDSA,
    }
    return key
}
```

我们可以看到，ecdsa.GenerateKey(crypto.S256()，rand）以太坊采用了椭圆曲线数字签名算法（ECDSA）生成一对公私钥，并选择的是secp256k1曲线。接下来本文问题的答案要揭晓了，

进入 PubkeyToAddress 一探究竟：

```
crypto/crypto.go

func PubkeyToAddress(p ecdsa.PublicKey) common.Address {
    pubBytes := FromECDSAPub(&p)
    return common.BytesToAddress(Keccak256(pubBytes[1:])[12:])
}

func Keccak256(data ...[]byte) []byte {
    d := sha3.NewKeccak256()
    for _, b := range data {
        d.Write(b)
    }
    return d.Sum(nil)
}
```

我们看到以太坊使用私钥通过 ECDSA算法推导出公钥，继而经过 Keccak-256 单向散列函数推导出地址。
至此，以太坊地址的生成过程，可以总结为下面 3 个步骤：
1. 创建随机私钥 (64 位 16 进制字符 / 32 字节)
2. 从私钥推导出公钥 (128 位 16 进制字符 / 64 字节)
3. 从公钥推导出地址 (40 位 16 进制字符 / 20 字节)
这是一件很奇妙的事情，这几行代码承载着亿万级别的资产，至简至美。以上的几行代码已经囊括密码学中大多数技术，比如随机数生成器、非对称加密，单向散列函数等。